

AD-A158 982

MICRO-COMPUTER BASED DYNAMIC ANALYSIS OF LINEAR  
UNDAMPED PLANE FRAME STRUCTURES(U) TEXAS A AND M UNIV  
COLLEGE STATION L GOSHORN AUG 85 N66314-73-A-2112

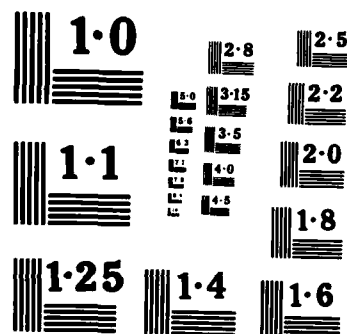
1/1

UNCLASSIFIED

F/G 13/13

NL

				END								
				FILED								
				DTL								



NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART

AD-A158 902

1

Micro-Computer Based Dynamic Analysis  
of  
Linear Undamped Plane Frame Structures

submitted to Dr. James Morgan  
in partial completion of the requirements  
for Degree of Master of Engineering

DTIC FILE COPY

DTIC  
SELECTED  
SEP 05 1985  
E

This document is available  
for publication and distribution  
without restriction.

by Larry Goshorn  
August 1985

85 9 03 134

AD NUMBER Lynch, J M.		DATE	DTIC ACCESSION NOTICE
1. REPORT IDENTIFYING INFORMATION			
A. ORIGINATING AGENCY NAVAL POSTGRADUATE SCHOOL, MONTEREY, CA 93943			
B. REPORT TITLE AND/OR NUMBER Reliability Modeling of Standby and Emergency Generating Systems			
C. MONITOR REPORT NUMBER by: J. Michael Lynch. June 1984 Texas A&M			
D. PREPARED UNDER CONTRACT NUMBER N66314-73-A-2112			<b>REQUESTER:</b> 1. Put your mailing address on reverse of form. 2. Complete Items 1 and 2. 3. Attach form to reports mailed to DTIC. 4. Use unclassified information only.
2. DISTRIBUTION STATEMENT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED.			

**DTIC:**

1. Assign AD Number.
2. Return to requester.

DTIC FORM 50  
(REF ID: A66314)

PREVIOUS EDITIONS ARE OBSOLETE



## Synopsis

The paper presents an assimilation of mathematical models and solutions needed in order to develop computer based analysis of dynamic structures. Using the variational formulation and a direct integration technique, a dynamic finite element model is developed. Modal analysis of unknown displacements of the structure, and the dynamic reduction of the structure are presented as alternative solutions. A system of micro-computer based programs which apply the presented solution techniques is described. The system of programs support varying cross sections of frame members, application of static, harmonic and non-harmonic loading conditions, and node displacements in the form of uniform base motion or independent node movement.

*Additional keywords: for charting; numerical integration. ←*

**N66314-73-A-2112**

Accession For	
NLS GRAB	
1985 100	
Unpublished	
Justification	
<i>form 50 per</i>	
By	
Distribution/	
Availability Codes	
Dist	
<b>A-1</b>	

This document has been approved for public release and its distribution is unlimited.



## Table of Contents

<u>Section</u>	<u>Page Number</u>
<b>Introduction</b> .....	1
<b>Finite Element Formulation</b> .....	2
<u>Discretization</u> .....	2
<u>Bar Element</u> .....	2
Variational Formulation of Governing Equation .....	3
Matrix Equations .....	3
Shape Functions .....	4
Element Matrices .....	4
<u>Beam Element</u> .....	5
Variational Formulation of Governing Equation .....	5
Matrix Equations .....	6
Shape Functions .....	6
Element Matrices .....	8
<u>Frame Element</u> .....	8
Description .....	8
Element Matrices .....	8
<u>Assembling Global Matrices</u> .....	9
<u>Application of Essential Boundary Conditions</u> .....	10
Independent Node Motion .....	10
Base Movement .....	10
<b>Time Approximations</b> .....	12
The Newmark method of Direct Integration .....	12
Procedure Summary .....	13
<b>Modal Analysis</b> .....	14
<u>The Eigenvalue Problem</u> .....	14
Uncoupling the Equations of Motion .....	15
<u>Solution of the Eigenvalue Problem</u> .....	15
Procedure Summary .....	17

<u>Section</u>	<u>Page Number</u>
<b>Reduction of the Equations of Motion</b> .....	18
Static Reduction .....	18
Including Inertial Effects .....	20
Procedure Summary .....	22
<b>The Dynamic Finite Element Program</b> .....	23
<u>DynFEP.menu</u> .....	23
<u>DynFEP.create data file</u> .....	23
<u>DynFEP.mass/stiffness</u> .....	24
<u>DynFEP.essential BC</u> .....	24
<u>DynFEP.reduce</u> .....	25
<u>DynFEP.eigen solver</u> .....	25
<u>DynFEP.uncouple/solve</u> .....	25
<u>DynFEP</u> .....	27
<u>Data Files</u> .....	27
Information Data File .....	28
Node Data File .....	28
Element Data File .....	29
Displacement History File .....	31
Reduction File .....	31
User defined force or displacement history files .....	32
Other Permanent data files .....	32
Temporary data files .....	33
<b>Conclusions</b> .....	34
<b>Bibliography</b> .....	36
<u>DynFEP.menu Flow Diagram and Listing</u> .....	A-1
<u>DynFEP.create data file Flow Diagram and Listing</u> .....	A-4
<u>DynFEP.mass/stiffness Flow Diagram and Listing</u> .....	A-7
<u>DynFEP.essential BC Flow Diagram and Listing</u> .....	A-11
<u>DynFEP.reduce Flow Diagram and Listing</u> .....	A-13
<u>DynFEP.eigen solver Flow Diagram and Listing</u> .....	A-16
<u>DynFEP.uncouple/solve Flow Diagram and Listing</u> .....	A-19
<u>DynFEP Flow Diagram and Listing</u> .....	A-24
<u>Listing of Universal Sub-Programs</u> .....	A-29

## Introduction

Computer analysis of dynamic structures has for some time been limited to mainframe computers. The importance of conducting a detailed analysis of any structure is evaluated against access to, and the cost of using a mainframe application to do that analysis. There are situations where analysis by mainframe is not possible or is not justified. In such cases, solution by hand may be impractical. There is a need to conduct rigorous analysis of dynamic structures that are too simple to justify using mainframe applications and too complicated to be solved by hand. Micro-computers are viewed as a possible means of satisfying this need.

The large amounts of memory required by the techniques which enable dynamic structures to be modeled in a form solvable by a digital computer have restricted their implementation on micro-computers. However, these techniques continue to be studied, refined, and combined with other techniques in the attempt to develop an optimal solution. In addition, micro-computers with abilities to address memory measured in the multi-megabytes are becoming widely available. With improved techniques and larger memory capacities, one can expect that rigorous analysis of simple dynamic structures will soon be done conveniently and inexpensively using micro-computers.

Toward that end, the mathematical formulations required to model dynamic structures on a micro-computer are synopsized. Combining these methods with a technique of reducing the complexity and number of resulting equations then results in an useful engineering analysis tool.

The paper first illustrates how the Finite Element Method is used to discretized the problem and express it in a matrix form. Next, the Newmark method of direct integration is used to simplify resulting integrations with respect to time. Further simplification of the equations are made possible through formulation and solution of the eigenvalue problem. Finally, a method for reducing the number of equations which must be solved is presented.

To show how the above techniques are applied to a micro-computer, a system of programs is described. The programs are capable of solving the resulting equations for dynamic analysis of undamped linear plane frame structures using any of the presented solutions. Flow diagrams and program listings are provided.



## Finite Element Formulation

The Finite Element method is widely used in the analysis of structures. It has the ability to systematically describe a structure in a matrix form which is easily applied to computer computation. Understanding the methods by which the matrix form is arrived at is important in understanding the capabilities and limitations of a computer application which employs the method.

The derivations presented in this section draw heavily from a text by J. N. Reddy, "An Introduction to Finite Element Method" (see the bibliography).

### Discretization

This model will describe a structure as an assemblage of two node frame elements. Each node will have three degrees of freedom, horizontal, vertical, and rotational movement. Mathematically, the frame element will consist of a superimposed one-dimensional bar element and a two-dimensional beam element. The bar and beam element are superimposed in a manner that assumes the transverse and rotational deflections/loads are independent from axial deflections/loads.

### Bar Element

The governing differential equation for the bar element is:

$$m \frac{\partial^2 u}{\partial t^2} + \frac{\partial}{\partial x} \left[ AE \frac{\partial u}{\partial x} \right] + F(x,t) = 0$$

Where  $F(x,t)$  is an axial forcing function which varies linearly with  $x$ ,  $m$  is the mass per unit length,  $A$  is the cross sectional area, and  $E$  is the modulus of elasticity. Damping has been ignored.

The variational formulation is found by integrating the governing equation against a test function over  $h$ , the length of a bar element.

$$\int_0^h v \left[ m \frac{\partial^2 u}{\partial t^2} - \frac{\partial}{\partial x} \left[ AE \frac{\partial u}{\partial x} \right] + F(t) \right] dx = 0$$

Integrating:

$$\int_0^h \left[ vm \frac{\partial^2 u}{\partial t^2} + AE \frac{\partial v}{\partial x} \frac{\partial u}{\partial x} + vF(x,t) \right] dx - vAE \frac{\partial u}{\partial x} \bigg|_{x=0}^{x=h} = 0$$

The last term of the above expression corresponds to the natural boundary conditions at either end of the bar element and will be denoted as  $P_1$ , the axial force on the left side and  $P_2$ , the axial force on the right side of the element.

Assume that  $u$  is interpolated by a linear expression of the form:

$$u = \sum_{j=1}^2 u_j(t) \psi_j(x)$$

Assuming that  $u$  and  $t$  can be separated, for any given time  $t > 0$ , the above expression is substituted for  $u$ , and  $v = \psi_i(x)$ . The matrix formulation results:

$$[M]\{u''\} + [K]\{u\} = \{F(t)\}$$

Where  $(')$  means differentiation with respect to  $t$  and:

$$M_{ij} = \int_0^h m \psi_i \psi_j dx$$

$$K_{ij} = \int_0^h AE \frac{d\psi_i}{dx} \frac{d\psi_j}{dx} dx$$

$$F_i = \int_0^h \psi_i F(x,t) dx + P_i(t)$$

Note that in the physical meaning of the above expressions,  $M_{ij}$   $K_{ij}$  do not vary with time. While  $F_i$  and  $P_i(t)$  vary with time, solutions will be based on the specific values of  $F(x,t)$  and  $P_i(t)$  at given points in time.

The interpolation functions  $\psi_i$  (for  $i=1$  to 2) must be sufficiently differentiable, independent of one another, complete, and must satisfy the essential boundary conditions. The expressions  $\psi_1 = a_1 + a_2x$  and  $\psi_2 = a_1 + a_2x$  are sufficiently differentiable, independent, complete, and values for the coefficients can be found to satisfy the essential boundary conditions. Below the Seredipity method is used to derive the interpolation functions:

The boundary conditions are:

$$\psi_1(x=0) = 1$$

$$\psi_1(x=h) = 0$$

$$\psi_2(x=0) = 0$$

$$\psi_2(x=h) = 1$$

Solving for coefficients:

$$\psi_1(0) = a_1 = 1$$

$$\psi_1(h) = 1 + a_2h = 0$$

$$\psi_2(h) = a_1 = 0$$

$$\psi_2(h) = a_2h = 1$$

$$\psi_1 = 1 + \frac{x}{h}$$

$$\psi_2 = \frac{x}{h}$$

These interpolation functions are used in the above expressions for  $M_{ij}$ ,  $K_{ij}$ , and  $F_i$  to derive the element matrices. In the derivation, the cross section of the element and the distributed force  $F(x,t)$ , are allowed to vary linearly with  $x$ . The modulus of elasticity, was assumed to be constant.

$$[K] = \frac{E}{2h} (A_1 + A_2) \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$[M] = \frac{h}{12} \begin{bmatrix} (3m_1 + m_2) & (m_1 + m_2) \\ (m_1 + m_2) & (m_1 + 3m_2) \end{bmatrix}$$

$$\{F\} = \frac{6}{h} \begin{Bmatrix} 2f_1 + f_2 \\ f_1 + 2f_2 \end{Bmatrix} + \begin{Bmatrix} P_1 \\ -P_2 \end{Bmatrix}$$

Where the subscripts indicate values at the left and right end of the bar element.

### The Beam Element

The governing equation for the beam element is

$$m \frac{\partial^2 u}{\partial t^2} + \frac{\partial^2}{\partial x^2} \left[ EI \frac{\partial^2 u}{\partial x^2} \right] + F(x,t) = 0$$

Where  $m$  is the mass per unit length,  $F(x,t)$  is a distributed transverse forcing function. Integrating the governing equation against a test function over the domain of the beam element gives

$$\int_0^h v \left[ m \frac{\partial^2 u}{\partial t^2} - \frac{\partial^2}{\partial x^2} \left[ EI \frac{\partial^2 u}{\partial x^2} \right] + F(x,t) \right] dx = 0$$

Integrating:

$$\int_0^h \left[ v m \frac{\partial^2 u}{\partial t^2} - \frac{\partial v}{\partial x} \frac{\partial}{\partial x} \left[ EI \frac{\partial^2 u}{\partial x^2} \right] + v F(x,t) \right] dx + v \frac{\partial}{\partial x} \left[ EI \frac{\partial^2 u}{\partial x^2} \right] \Bigg|_{x=0}^{x=h} = 0$$

The last term of the above expression corresponds to the shear (natural boundary condition) at either end of the element and will be denoted as  $Q_1$  (shear at the left end) and  $Q_3$  (shear at the right end).

Integrating the second term again:

$$\int_0^h \left[ v m \frac{\partial^2 u}{\partial t^2} + EI \frac{\partial^2 v}{\partial x^2} \frac{\partial^2 u}{\partial x^2} + v F(x,t) \right] dx + v Q \Bigg|_{x=0}^{x=h} - EI \frac{\partial v}{\partial x} \frac{\partial^2 u}{\partial x^2} \Bigg|_{x=0}^{x=h} = 0$$

The last term of the above expression corresponds to the moment at either end of the element (natural boundary condition), and will be denoted as  $Q_2$  (moment at the left end) and  $Q_4$  (moment at the right end).

The displacement is again interpolated by an expression of the form:

$$u = \sum_{j=1}^2 u_j(t) \psi_j(x)$$

Substituting the above expression for  $u$ , and  $v = \psi_i(x)$  results in the matrix formulation

$$[M]\{u''\} + [K]\{u\} = \{F(t)\}$$

Where (') means differentiation with respect to  $t$  and

$$M_{ij} = \int_0^h m \psi_i \psi_j dx$$

$$K_{ij} = \int_0^h AE \frac{d^2 \psi_i}{dx^2} \frac{d^2 \psi_j}{dx^2} dx$$

$$F_i = \int_0^h \psi_i F(x,t) dx + Q_i(t)$$

The interpolation functions  $\psi_i$  (for  $i=1$  to  $4$ ) must be sufficiently differentiable, independent of one another, complete, and must satisfy the essential boundary conditions. The expressions  $\psi_1 = a_1 + a_2x + a_3x^2 + a_4x^3$  and  $\psi_2 = a_1 + a_2x + a_3x^2 + a_4x^3$  are sufficiently differentiable, independent, complete, and values for the coefficients can be found to satisfy the essential boundary conditions. Below the Seredipity method is used to derive the interpolation functions.

The boundary conditions ( ' denotes differentiation with respect to  $x$ ):

$$\psi_1(x=0) = 1$$

$$\psi_2(x=0) = 0$$

$$\psi_1'(x=0) = 0$$

$$\psi_2'(x=0) = -1$$

$$\psi_1(x=h) = 0$$

$$\psi_2(x=h) = 0$$

$$\psi_1'(x=h) = 0$$

$$\psi_2'(x=h) = 0$$

Solving for the coefficients:

$$\psi_1(0) = a_1 = 1$$

$$\psi_2(0) = a_1 = 0$$

$$\psi_1'(0) = a_2 = 0$$

$$\psi_2'(0) = a_2 = -1$$

$$\psi_1(h) = 1 + a_3h^2 + a_4h^3 = 0$$

$$\psi_2(h) = -h + a_3h^2 + a_4h^3 = 0$$

$$\psi_1'(h) = 2a_3h + 3a_4h^2 = 0$$

$$\psi_2'(h) = -1 + 2a_3h + 3a_4h^2 = 0$$

$$\begin{bmatrix} h^2 & h^3 \\ 2h & 3h^2 \end{bmatrix} \begin{Bmatrix} a_3 \\ a_4 \end{Bmatrix} = \begin{Bmatrix} -1 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} h^2 & h^3 \\ 2h & 3h^2 \end{bmatrix} \begin{Bmatrix} a_3 \\ a_4 \end{Bmatrix} = \begin{Bmatrix} h \\ 1 \end{Bmatrix}$$

$$\psi_1 = 1 - 3\frac{x^2}{h^2} + 2\frac{x^3}{h^3}$$

$$\psi_2 = -x + 2\frac{x^2}{h} - \frac{x^3}{h^2}$$

The boundary conditions ( ' denotes differentiation with respect to x):

$$\psi_3(x=0) = 0$$

$$\psi_4(x=0) = 0$$

$$\psi_3'(x=0) = 0$$

$$\psi_4'(x=0) = 0$$

$$\psi_3(x=h) = 1$$

$$\psi_4(x=h) = 0$$

$$\psi_3'(x=h) = 0$$

$$\psi_4'(x=h) = -1$$

Solving for the coefficients:

$$\psi_3(0) = a_1 = 0$$

$$\psi_4(0) = a_1 = 0$$

$$\psi_3'(0) = a_2 = 0$$

$$\psi_4'(0) = a_2 = 0$$

$$\psi_3(h) = a_3h^2 + a_4h^3 = 1$$

$$\psi_4(h) = a_3h^2 + a_4h^3 = 0$$

$$\psi_3'(h) = 2a_3h + 3a_4h^2 = 0$$

$$\psi_4'(h) = 2a_3h + 3a_4h^2 = -1$$

$$\begin{bmatrix} h^2 & h^3 \\ 2h & 3h^2 \end{bmatrix} \begin{Bmatrix} a_3 \\ a_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}$$

$$\begin{bmatrix} h^2 & h^3 \\ 2h & 3h^2 \end{bmatrix} \begin{Bmatrix} a_3 \\ a_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -1 \end{Bmatrix}$$

$$\psi_3 = 3\frac{x^2}{h^2} - 2\frac{x^3}{h^3}$$

$$\psi_4 = \frac{x^2}{h} - \frac{x^3}{h^2}$$

These interpolation functions are used in the above expressions for  $M_{ij}$ ,  $K_{ij}$ , and  $F_{ij}$  to derive the beam element matrices. The cross section of the element and the transverse loading function  $F(x,t)$  are allowed to vary linearly, but the modulus of elasticity is held constant.

$$[K] = \frac{E}{h^3} \begin{bmatrix} 6(I_1 + I_2) & -h(4I_1 + 2I_2) & -6(I_1 + I_2) & -h(2I_1 + 4I_2) \\ & h^2(3I_1 + I_2) & h(4I_1 + 2I_2) & h^2(I_1 + I_2) \\ (sym) & & 6(I_1 + I_2) & h(2I_1 + 4I_2) \\ & & & h^2(I_1 + 3I_2) \end{bmatrix}$$

$$[M] = \frac{1}{830} \begin{bmatrix} h(10m_1 + 3m_2) & -h^2(15I_1 + 7I_2) & h(9m_1 + 9m_2) & h^2(7m_1 + 6m_2) \\ & h^3(5m_1 + 3m_2) & -h^2(6m_1 + 7m_2) & -h^3(m_1 + m_2) \\ (sym) & & h(3m_1 + 10m_2) & h^2(7m_1 + 15m_2) \\ & & & h^3(3m_1 + 5m_2) \end{bmatrix}$$

$$\{F\} = \frac{h}{60} \begin{bmatrix} 15(f_1 - 3f_2) \\ -h(3f_1 + 2f_2) \\ 3(3f_1 + 7f_2) \\ h(2f_1 + 3f_2) \end{bmatrix} + \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}$$

### Frame Element

The bar and beam elements are now superimposed upon one another to form the frame element. It is assumed that forces and displacements in the axial direction and in the transverse direction are independent of one another. The resulting element matrices are shown below.

$$[K] = \frac{E}{2h^3} \begin{bmatrix} h^2(A_1 + A_2) & & & & -h^2(A_1 + A_2) \\ & 12(I_1 + I_2) & -2h(4I_1 + 2I_2) & & -12(I_1 + I_2) & -2h(2I_1 + 4I_2) \\ & & 2h^2(3I_1 + I_2) & & 2h(4I_1 + 2I_2) & 2h^2(I_1 + I_2) \\ (sym) & & & h^2(A_1 + A_2) & & \\ & & & & 12(I_1 + I_2) & 2h(2I_1 + 4I_2) \\ & & & & & 2h^2(I_1 + I_2) \end{bmatrix}$$

$$[M] = \frac{1}{840} \begin{bmatrix} 70h(3m_1+m_2) & & & & 70h(m_1+m_2) \\ & 24h(10m_1+3m_2) & -2h^2(15m_1+7m_2) & & 54h(m_1+m_2) & 2h^2(7m_1+6m_2) \\ & & h^3(5m_1+3m_2) & & -2h^2(6m_1+7m_2) & -3h^3(m_1+m_2) \\ & & & 70h(m_1+m_2) & & \\ & (sym) & & & 24h(3m_1+10m_2) & 2h^2(7m_1+15m_2) \\ & & & & & h^3(3m_1+5m_2) \end{bmatrix}$$

$$\{F\} = \frac{1}{60} \begin{bmatrix} 10h(2f_1 + f_2) \\ 15h(f_1 - 3f_2) \\ -h^2(3f_1 + 2f_2) \\ 10h(f_1 + 2f_2) \\ 3h(3f_1 + 7f_2) \\ h^2(2f_1 + 3f_2) \end{bmatrix} + \begin{bmatrix} P_1 \\ Q_1 \\ Q_2 \\ -P_2 \\ Q_3 \\ Q_4 \end{bmatrix}$$

### Assembling Global Matrices

Prior to assembling the element matrices into the global matrices, the element local coordinates must be converted to global coordinates. This is done by premultiplying the stiffness and mass matrices with the following transformation matrix. The element force matrix is premultiplied by the transpose of the transformation matrix. The angle  $\theta$  is measured from the from the positive x direction clockwise:

$$\begin{bmatrix} \cos\theta & -\sin\theta & & & & \\ & \cos\theta & & & & \\ & & 1 & & & \\ & & & \cos\theta & -\sin\theta & \\ (sym) & & & & \cos\theta & \\ & & & & & 1 \end{bmatrix}$$

When the element matrices are assembled, the internal element forces,  $P_1$ ,  $Q_1$ ,  $Q_2$ ,  $P_2$ ,  $Q_3$ , and  $Q_4$  are canceled out by the internal element forces of adjoining elements. There may, however, be externally applied loads at the nodes, if this



is the case, they are added into the formulation as shown in the above expression for the force matrix. Note however, that since the loads are applied directly to the nodes, that the coordinate transformation is not appropriate.

#### Applying Essential Boundary Conditions

In the development of the mass and stiffness matrices, the shape functions were developed in order to account for essential boundary conditions but essential boundary conditions were never actually applied. As a result, the stiffness matrix is currently singular and can not be inverted (ie, the problem can not be solved as is). One consequence of this is that this configuration can not be used to solve for displacements of structures which are not anchored in some way to an immovable object (as an example an object floating in space). Application of essential boundary conditions constrain the structure and the stiffness matrix becomes non-singular.

There are two approaches to apply the essential boundary conditions. Since the displacement of a node in a particular degree of freedom is known, the corresponding equation in the matrix formulation is simply changed to reflect the known value. The Gauss elimination scheme used to solve the simultaneous equations will insure that the influence of the displaced node is properly reflected thorough out the structure. This is the method used in the program DynFEP. It has the advantage that all constrained nodes need not all move at once or in the same directions, in addition rotations of individual nodes can be investigated with this approach.

An alternate approach is described in the referenced text by Clough & Penzien. The common approach used in earthquake analysis, is to drop the row and column corresponding to the displaced/constrained node from the formulation, reducing the number of simultaneous equations to be solved. Then effects of base motion are added into the formulation. This is done by adopting a coordinate system where the unknown displacements are relative to the movement of the base of the structure. Then an inertial term is added to the right hand side of appropriate equations. As an example, if the base of the structure experienced a horizontal displacement, then an inertia term would be added to every equation in the matrix formulation which pertained to horizontal displacements. In matrix formulation an acceleration vector accounting for horizontal and vertical movement is developed and premultiplied by the mass matrix to obtain the inertia term, this column matrix is then added to the right hand side of the

equations.

Rotations are normally disregarded in this approach. First because earthquakes seldom display any rotational components and second because the bookkeeping chore is very burdensome. The inertia effect of a node rotation on another node is proportional to the lever arm between the two nodes. Thus, for each node that rotates the lever arm between it and all other nodes must be found in calculating the inertia term. In addition, its very difficult to conceptualize the inertial effects of one node on another when several nodes are rotating.

The DynFEP.uncouple/solve and DynFEP.reduce programs presented below are formulated in the above manner.

## Time Approximations

The Finite Element Method provides a method of converting the differentials with respect to  $x$  in the governing equations into a linear algebra problem suitable for solution by computer. During the derivation it was assumed that displacements with respect to space and time could be separated. We are now faced with solving the resulting matrix differential equation in time.

$$[M]\{u''\} + [K]\{u\} = \{F(t)\}$$

In order to utilize a computer based solution, the above differential equation must also be simplified to an algebraic form. The Newmark method of direct integration is a commonly used technique to accomplish this. The Newmark method is described in referenced texts by Reddy, Clough & Penzien, and Bathe & Wilson. It is based on the following assumptions:

$$\{u'\}_{t+\Delta t} = \{u'\}_t + [(1 - \delta)\{u''\}_t + \delta\{u''\}_{t+\Delta t}]\Delta t \quad (1)$$

$$\{u\}_{t+\Delta t} = \{u\}_t + \{u'\}_t\Delta t + [(1/2 - \alpha)\{u''\}_t + \alpha\{u''\}_{t+\Delta t}]\Delta t^2 \quad (2)$$

Where  $\alpha$  and  $\delta$  are parameters that can control the integration accuracy and stability. When  $\delta=1/2$  and  $\alpha=1/6$  the above expressions correspond to a linear acceleration assumption. When  $\delta=1/2$  and  $\alpha=1/4$  above expressions correspond to a constant-average-acceleration assumption.

Working with equation (2), acceleration for a new time increment can be expressed in terms of current displacement and values from the last time increment.

$$\alpha\{u''\}_{t+\Delta t}\Delta t = \{u\}_{t+\Delta t} - \{u\}_t - \{u'\}_t\Delta t - (1/2 - \alpha)\{u''\}_t\Delta t$$

$$\{u''\}_{t+\Delta t} = \frac{1}{\alpha\Delta t^2} (\{u\}_{t+\Delta t} - \{u\}_t) - \frac{1}{\alpha\Delta t} \{u'\}_t - \frac{(1/2 - \alpha)}{\alpha} \{u''\}_t$$

$$\{u''\}_{t+\Delta t} = a_1(\{u\}_{t+\Delta t} - \{u\}_t) - a_2\{u'\}_t - a_3\{u''\}_t \quad (3)$$

Substituting equation (3) into the discretized equations of motion:

$$\begin{aligned} [M](a_1(\{u\}_{t+\Delta t} - \{u\}_t) - a_2\{u'\}_t - a_3\{u''\}_t) + [K]\{u\}_{t+\Delta t} &= \{F\}_{t+\Delta t} \\ (a_1[M] + [K])\{u\}_{t+\Delta t} &= \{F\}_{t+\Delta t} + [M](\{u\}_t + a_2\{u'\}_t + a_3\{u''\}_t) \end{aligned} \quad (4)$$

Using the above equation the procedure for direct integration is as follows:

- 1) Knowing displacement, velocity, and acceleration from the last time step (or from initial conditions), find displacements for next time step using equation (4) above.
- 2) Using equation (3) find current acceleration.
- 3) Using equation (1) find current velocity.
- 4) Proceed to next time step.

The Newmark method is unconditionally stable for  $\alpha=1/2$  and  $\delta=1/4$  and is normally stable for  $\alpha=1/2$  and  $\delta=1/6$ . In order to also insure accuracy of the method,  $\Delta t$  should not exceed:

$$\Delta t_{\max} = \frac{T_{\min}}{\pi} = \frac{2}{\omega_{\max}}$$

## Modal Analysis

The Finite Element Method and the Newmark method, are used above to convert the differentials which govern movement of plane frame structures to a set of simultaneous algebraic equations. These equations are then solved repeatedly in small time steps to obtain the displacement response of the structure over time. Given this method of solution it should be obvious that any means to further simplify the solution process will be valuable.

The texts by Clough & Penzien, and Bathe & Wilson present a widely used method to uncouple the simultaneous equations so that they may be solved independently of one another. The method involves expressing the equations of motion as an eigenvalue problem, solving the eigenvalue problem, and then re-expressing the equations of motion in a coordinate system which has been generalized by the eigen vectors.

### The Eigenvalue Problem

If the structure in question is in free vibration then the forces on the right hand side of the equations of motion are equal to zero,  $[M]\{u''\} + [K]\{u\} = \{0\}$ . The solution for each degree of freedom is then  $\{u\} = \{\phi\}\sin(\omega t)$ . Substituting this solution into the equations of motion

$$-\omega^2 [M]\{\phi\}\sin(\omega t) + [K]\{\phi\}\sin(\omega t) = \{0\}$$

$$([K] - \omega^2 [M])\{\phi\}\sin(\omega t) = \{0\}$$

Since  $\sin(\omega t)$  is not equal to zero for all  $t$ ,

$$([K] - \omega^2 [M])\{\phi\} = \{0\}$$

A non-trivial solution to this system of simultaneous equations exists only when  $|[K] - \omega^2 [M]| = 0$ . When this determinate is expanded, it results in an algebraic equation of the  $n^{\text{th}}$  degree (where the dimensions of  $[K]$  and  $[M]$  are  $n$ -by- $n$ ). The  $n$  roots to this equation,  $\omega_i$  (where  $i=1, 2, \dots, n$ ), represent the frequencies of the  $n$  modes of vibration that are possible in the system. The

associated eigen vectors,  $\{\phi_i\}$ , describe the relative displacements of the structure nodes in the  $i^{\text{th}}$  response mode. The total response is given by the sum of the mode responses each multiplied by a currently unknown amplitude.

The eigen vectors are  $[K]$  and  $[M]$ -orthogonal. Thus  $\{\phi_i\}^T [M] \{\phi_i\} = [M_n]$ , where  $[M_n]$  is a diagonal matrix, and  $\{\phi_i\}^T [K] \{\phi_i\} = [K_n]$ , where  $[K_n]$  is a diagonal matrix. In addition  $\{\phi_i\}^T [M] \{\phi_j\} = [0]$ , and  $\{\phi_i\}^T [K] \{\phi_j\} = [0]$  where  $i \neq j$ .

The advantage of the modal analysis is seen when a generalized coordinate system is defined as  $\{u\} = [\phi] \{\xi\}$ . Where  $[\phi]$  is a matrix made up of the individual eigen vectors. Premultiplying the original equations of motion by  $[\phi]^T$  and substituting the generalized coordinate system into the equation of motion results in:

$$[\phi]^T [M] [\phi] \{\xi''\} + [\phi]^T [K] [\phi] \{\xi\} = [\phi]^T \{F(t)\}$$

$$[M_n] \{\xi''\} + [K_n] \{\xi\} = [\phi]^T \{F(t)\}$$

Stated in terms of the generalized coordinate system, the equations of motion, are uncoupled. Since  $[M_n]$  and  $[K_n]$  are diagonal matrices each equation in the above system of equations is independent of the others.

#### Solution of the Eigenvalue Problem

Clough & Penzien describe a matrix iteration method originally developed by Stodola to solve the eigenvalue problem. The eigenvalue problem is restated as follows:

$$[K] \{\phi\} = \omega^2 [M] \{\phi\}$$

Rearranging:

$$[K]^{-1} [M] \{\phi\} = \frac{1}{\omega^2} \{\phi\}$$

The Stodola method consists of using a guessed trial mode shape,  $\{\phi_{\text{trial}}\}$ , on the left-hand side of the above equation to calculate a new guess on the

right-hand side. The square of the frequency is obtained by dividing any component of the new guess by the same component of the original guess. The new guess will always be better than the old guess, and the process will converge to the lowest mode or frequency.

Using the orthogonal properties of the eigenvectors it is possible to eliminate the components of any particular mode from the total response of the structure. By eliminating the first mode components from the total response it is possible to use the above method to find the second mode response (since it would now be the lowest). Extending this approach, succeeding modes can also be found.

Expressing a trial mode shape in terms of its modal components and then premultiplying both sides by  $\{\phi_1\}^T[M]$

$$\{\phi_{\text{trial}}\} = \sum_{i=1}^n \{\phi_i\} A_i = \{\phi_1\} A_1 + \{\phi_2\} A_2 + \{\phi_3\} A_3 + \cdots + \{\phi_n\} A_n$$

$$\{\phi_1\}^T[M]\{\phi_{\text{trial}}\} = \{\phi_1\}^T[M]\{\phi_1\} A_1 + \{\phi_1\}^T[M]\{\phi_2\} A_2 + \cdots + \{\phi_1\}^T[M]\{\phi_n\} A_n$$

$$\{\phi_1\}^T[M]\{\phi_{\text{trial}}\} = \{\phi_1\}^T[M]\{\phi_1\} A_1$$

Solving for  $A_1$ :

$$A_1 = \frac{\{\phi_1\}^T[M]\{\phi_{\text{trial}}\}}{\{\phi_1\}^T[M]\{\phi_1\}}$$

Subtracting the first mode shape from the original trial mode shape results in a new trial with no first mode components,  $\{\phi_{\text{trial}(1)}\}$ .

$$\{\phi_{\text{trial}(1)}\} = \{\phi_{\text{trial}}\} - \{\phi_1\} A_1 = \{\phi_{\text{trial}}\} - \frac{\{\phi_1\} \{\phi_1\}^T[M]}{\{\phi_1\}^T[M]\{\phi_1\}} \{\phi_{\text{trial}}\}$$

This can also be expressed as  $\{\phi_{\text{trial}(1)}\} = [S_1]\{\phi_{\text{trial}}\}$ , where:

$$[S_1] = [I] - \frac{\{\phi_1\} \{\phi_1\}^T[M]}{\{\phi_1\}^T[M]\{\phi_1\}}$$

The  $[S_1]$  matrix is referred to as the first mode sweeping matrix. It has the property that when multiplied by any trial vector it removes the first-mode component. Sweeping matrices which remove more than one mode shape can be constructed in a similar matter. As an example, a sweeping matrix which will remove the first, second, and third mode shape components from a trial vector would be constructed as follows:

$$[S_3] = [I] - \frac{\{\phi_1\}\{\phi_1\}^T[M]}{\{\phi_1\}^T[M]\{\phi_1\}} - \frac{\{\phi_2\}\{\phi_2\}^T[M]}{\{\phi_2\}^T[M]\{\phi_2\}} - \frac{\{\phi_3\}\{\phi_3\}^T[M]}{\{\phi_3\}^T[M]\{\phi_3\}}$$

The resulting Stodola matrix iteration model to find the fourth mode shape and eigenvalue becomes:

$$[K]^{-1} [M][S_3]\{\phi\} = \frac{1}{\omega^2} \{\phi\}$$

The method of solution suggested by the above methods consists of the following:

- 1) Find lowest mode shape and eigenvalue using the matrix iteration.
- 2) Using the newly calculated first mode shape eliminate the first mode components.
- 3) Repeat the procedure for the next mode shape and eigenvalue.

Each successive mode shape is based on eliminating the previous mode's components. According to Clough & Penzien, numerical roundoff errors which allow any previous mode components to remain in the sweeping matrix are accumulative. Thus in order for the sweeping matrix to perform effectively for higher modes it is necessary to retain a great deal of precision in calculating the lower modes.

The eigenvalues and eigen vectors are now used to form the uncoupled equations of motion. The Newmark method is applied to the resulting independent equations. The independent equations are solved in the terms of the generalized coordinates,  $\{\xi\}$ , while stepping through time. In each time step the real displacement vectors are found from the relation  $\{u\} = [\Phi]\{\xi\}$ .



## Reduction of the Equations of Motion

When the above methods are applied to real structures, very large matrices and correspondingly large computer capacity are required to solve the resulting equations. It is thus desirable to reduce the number of equations which must be solved. In the study of structures it has been determined that only the first few response modes contribute significantly to the overall response a structure. It is therefore reasonable to ignore the higher modes of response if it will reduce the number of equations to be solved.

Robert J. Guyan described such a method of reducing the number of equations to be solved in a paper to the AIAA Journal. The method consists of a static reduction of the structure. Working with the static description of the structure, the matrices are partitioned by the nodes which will be retained in the solution (referred to as the primary nodes), and the nodes which will be eliminated from the formulation (referred to as the secondary nodes). It is assumed that no external loads will be applied to the secondary nodes.

$$\begin{bmatrix} [K_{pp}] & [K_{ps}] \\ [K_{sp}] & [K_{ss}] \end{bmatrix} \begin{Bmatrix} \{u_p\} \\ \{u_s\} \end{Bmatrix} = \begin{Bmatrix} \{f_p\} \\ \{f_s\} \end{Bmatrix}$$

This results in the following two matrix equations:

$$[K_{pp}]\{u_p\} + [K_{ps}]\{u_s\} = \{f_p\} \quad (5)$$

$$[K_{sp}]\{u_p\} + [K_{ss}]\{u_s\} = \{f_s\} = \{0\} \quad (6)$$

Multiplying the second equation by  $[K_{ps}][K_{ss}]^{-1}$

$$[K_{ps}][K_{ss}]^{-1} [K_{sp}]\{u_p\} + [K_{ps}][K_{ss}]^{-1} [K_{ss}]\{u_s\} = \{0\}$$

$$[K_{ps}][K_{ss}]^{-1} [K_{sp}]\{u_p\} + [K_{ps}]\{u_s\} = \{0\} \quad (7)$$

Subtracting equation (7) from equation (5) gives:

$$[K^*]\{u_p\} = \{f_p\}$$

Where  $[K^*]$  is the reduced stiffness matrix, found by the following expression:

$$[K^*] = [K_{pp}] - [K_{ps}][K_{ss}]^{-1}[K_{sp}]$$

In addition, from equation (7) a transformation matrix can be obtained to convert between the primary and secondary displacement values.

$$[K_{ps}]\{u_s\} = -[K_{ps}][K_{ss}]^{-1}[K_{sp}]\{u_p\}$$

$$\{u_s\} = -[K_{ss}]^{-1}[K_{sp}]\{u_p\} = -[T]\{u_p\} \quad (8)$$

Rearranging:

$$\{u\} = \begin{Bmatrix} [I] \\ -[T] \end{Bmatrix} \begin{Bmatrix} \{u_p\} \\ \{u_p\} \end{Bmatrix}$$

The kinetic energy of the structure can be expressed as:

$$T = 1/2 \{u'\}^T [M] \{u'\}$$

Substituting the above expression for  $\{u\}$  results in:

$$T = 1/2 \{u'_p\}^T \begin{Bmatrix} [I] \\ -[T] \end{Bmatrix}^T \begin{bmatrix} [M_{pp}] & [M_{ps}] \\ [M_{sp}] & [M_{ss}] \end{bmatrix} \begin{Bmatrix} [I] \\ -[T] \end{Bmatrix} \{u'_p\}$$

Thus it can be seen that the reduced  $[M]$  can be expressed as:

$$[M^*] = \begin{Bmatrix} [I] \\ -[T] \end{Bmatrix}^T \begin{bmatrix} [M_{pp}] & [M_{ps}] \\ [M_{sp}] & [M_{ss}] \end{bmatrix} \begin{Bmatrix} [I] \\ -[T] \end{Bmatrix}$$

Expanding the above expression and substituting the expression for  $[T]$  we obtain the simplified expression for  $[M^*]$ :

$$[M^*] = [M_{pp}] - [M_{ps}][K_{ss}]^{-1}[K_{sp}] - [K_{ps}][K_{ss}]^{-1}([M_{sp}] - [M_{ss}][K_{ss}]^{-1}[K_{sp}]) \quad (9)$$

The above method has reduced the mass and stiffness matrix of the structure and therefore the number of equations which must be solved. However, the transformation matrix  $[T]$  used to find the displacement of secondary nodes has ignored any inertial effects. The exact expression for  $[T]$  is found by expressing the eigenvalue problem in the partitioned form:

$$\begin{bmatrix} [K_{pp}] & [K_{ps}] \\ [K_{sp}] & [K_{ss}] \end{bmatrix} \begin{Bmatrix} \{\varphi_p\} \\ \{\varphi_s\} \end{Bmatrix} - \omega^2 \begin{bmatrix} [M_{pp}] & [M_{ps}] \\ [M_{sp}] & [M_{ss}] \end{bmatrix} \begin{Bmatrix} \{\varphi_p\} \\ \{\varphi_s\} \end{Bmatrix} = \{0\}$$

Working with the second partitioned matrix equation the exact transformation matrix for the eigen vectors is obtained:

$$[K_{sp}]\{\varphi_p\} + [K_{ss}]\{\varphi_s\} - \omega^2[M_{sp}]\{\varphi_p\} - \omega^2[M_{ss}]\{\varphi_s\} = \{0\}$$

$$(\omega^2[M_{ss}] - [K_{ss}])\{\varphi_s\} = (\omega^2[M_{sp}] - [K_{sp}])\{\varphi_p\}$$

$$[T] = (\omega^2[M_{ss}] - [K_{ss}])^{-1}(\omega^2[M_{sp}] - [K_{sp}])$$

Note that if the inertial terms in the above expression are neglected, the same transformation matrix developed earlier, based on a static derivation, is obtained. The above expression, however, involves an eigenvalue based on the complete set of equations and requires that an inversion of the  $(\omega^2[M_{ss}] - [K_{ss}])$  term be found for each eigenvalue.

Charles Miller describes a transformation matrix which is more accurate than the one derived previously and more convenient than the exact formulation in a paper to the Journal of the Structural Division, Proceedings of the ASCE.

Mr. Miller notes that  $\omega^2[M_{ss}]$  and  $\omega^2[M_{sp}]$  are normally small when compared to  $[K_{ss}]$  and  $[K_{sp}]$ . With this in mind he expands the  $(\omega^2[M_{ss}] - [K_{ss}])^{-1}$  term of the exact formulation about  $[K_{ss}]^{-1}$  dropping the  $\omega^4$  terms in comparison to  $\omega^2$  terms. This results in a revised  $[T]$ :

$$[T] = [K_{ss}]^{-1} [K_{sp}] + \omega^2(-[K_{ss}]^{-1} [M_{sp}] + [K_{ss}]^{-1} [M_{ss}][K_{ss}]^{-1} [K_{sp}])$$

Expressing the first equation of the partitioned eigenvalue problem in terms of  $\{\phi_p\}$

$$[K_{pp}]\{\phi_p\} + [K_{ps}][T]\{\phi_p\} - \omega^2[M_{pp}]\{\phi_p\} - \omega^2[M_{ps}][T]\{\phi_p\} = \{0\}$$

Expanding the  $\omega^2[M_{ps}][T]\{\phi_p\}$  term again dropping the  $\omega^4$  terms

$$\omega^2[M_{ps}][T]\{\phi_p\} = -\omega^2[M_{ps}][K_{ss}]^{-1} [M_{sp}]\{\phi_p\}$$

Expanding the  $[K_{ps}][T]\{\phi_p\}$  term

$$[K_{ps}][T]\{\phi_p\} = ([K_{ps}][K_{ss}]^{-1} [K_{sp}] - \omega^2[K_{ps}](-[K_{ss}]^{-1} [M_{sp}] + [K_{ss}]^{-1} [M_{ss}][K_{ss}]^{-1} [K_{sp}])\{\phi_p\}$$

Substituting these expanded expressions into the eigenvalue problem:

$$([K_{pp}] + [K_{ps}][K_{ss}]^{-1} [K_{sp}] - \omega^2[K_{ps}](-[K_{ss}]^{-1} [M_{sp}] + [K_{ss}]^{-1} [M_{ss}][K_{ss}]^{-1} [K_{sp}] - \omega^2[M_{pp}] + \omega^2[M_{ps}][K_{ss}]^{-1} [M_{sp}])\{\phi_p\} = \{0\}$$

Rearranging gives:

$$([K_{pp}] - [K_{ps}][K_{ss}]^{-1}[K_{sp}])\{\phi_p\} = \omega^2([M_{pp}] - [M_{ps}][K_{ss}]^{-1}[K_{sp}] - [K_{ps}][K_{ss}]^{-1}([M_{sp}] - [M_{ss}][K_{ss}]^{-1}[K_{sp}])\{\phi_p\})$$

$$[K^*]\{\phi_p\} = \omega^2[M^*]\{\phi_p\}$$

The revised transformation matrix results in the same expressions for the reduced mass and stiffness matrices. The revised transformation matrix requires only one inverse of a partition of the stiffness matrix, not a new inverse for each eigenvalue. In addition, the revised transformation matrix should provide more accurate displacements since it partly accounts for inertia terms. The more accurate displacements provide a more accurate basis for approximating internal forces.

The process suggested by the above formulation proceeds as follows:

- 1) Partition mass and stiffness matrices and find reduced matrices using equations (8) and (9).
- 2) Solve the eigenvalue problem for reduced mass and stiffness matrices.
- 3) For each eigenvalue find the transformation matrix.
- 4) Use each transformation matrix to obtain full eigen vector.

Once the full eigen vectors are found the solution proceeds the same as under Modal Analysis. It should be noted, however that the dimensions of various matrices have been changed.

Where  $n$  equals the number of unknowns in the structure, and  $m$  equals the number of modes retained in the solution, the dimensions of the eigenvalue matrix is  $m \times m$  and the dimension of the complete eigen vector matrix  $[\Phi]$  is  $n \times m$ . When the generalized mass matrix is found from the relation  $[\Phi]^T[M][\Phi]$ , its dimensions are  $m \times m$ . Due to the orthogonality of the eigen vector matrix, the generalized mass matrix is still diagonal and there remain only  $m$  independent equations to be solved. Converting the generalized solutions to real coordinates using the relation  $\{u\} = [\Phi]\{\xi\}$  results in full size displacement matrix ( $[\Phi]$  is dimensioned  $n \times m$  and  $\{\xi\}$  is dimensioned  $m \times 1$ ).

## **The Dynamic Finite Element Program**

The Dynamic Finite Element Program is a system of programs developed to apply the methods presented above. The programs are written in Micro-Soft Basic for the Apple Macintosh, version 2 (Micro-Soft Inc. is currently developing versions of this advanced version of Basic for IBM compatible machines). Flow diagrams and listings of the programs are provided in Appendix A. A description of each program and its operation follows.

### DynFEP.menu

The DynFEP.menu program serves to connect the system of programs together. It provides a menu from which the user can choose to create data files which describe structures, or to solve problems which have been defined earlier. Problems may be solved in any of three ways, using dynamic reduction, using modal analysis, or direct numerical integration of the equations of motion using the Newmark method.

The DynFEP.menu program maintains control of the flow of execution by passing five variables to each program in the system. The five variables are: the number of global nodes labeled as GN; the number of elements labeled as NE; the number of unknowns remaining after application of the essential boundary conditions labeled as n (if essential boundary conditions are applied by reducing the number of equations); the number of modes to be retained in the solution labeled m (if the structure is to be reduced); and a string variable describing the chosen solution method labeled as Path\$.

### DynFEP.create data file

This program creates the data files which describe the structure, and forcing functions and displacements applied against it. Data describing the structure is entered using Basic DATA statements. A separate Basic program listing containing the only the desired DATA statements is prepared and saved under ASCII (text only) format. The program assumes that such a program has been prepared and merges with it. When the resulting new program is executed, it will read the prepared data and create the required structure data files.

The DATA statements must be formatted to match the READ statements in the DynFEP.create program. This is normally accomplished by copying a previously created set of DATA statements, and modifying them to fit the new problem using Basic's editing capabilities.

#### DynFEP.mass/stiffness

This program reads previously created structure data files and assembles the global stiffness matrix, the global mass matrix, and the global static forces matrix. The program steps through the structure elements, and constructs the element matrices using the relations presented above. The orientation of the element is checked and the matrices are transformed if necessary. Then the element matrices are assembled into the global matrices in accordance with their end node points.

The program will be executed as determined necessary by the DynFEP.menu program. The above matrices will be assembled once for any particular structure, it is not necessary to reassemble the global matrices for different applied dynamic forces or specified displacements. When the program completes assemblage of the global matrices it will check a specified solution-pathway which was set by the DynFEP.menu program. There are two possible paths, directly to DynFEP if non-modal analysis is to be done, or to DynFEP.essential BC if modal analysis is to be done. The program will chain to the appropriate program.

#### DynFEP.essential BC

This program reads the structure node information file, determines where essential boundary conditions are to be applied, and then applies the conditions by eliminating the appropriate rows and columns of the global mass, stiffness, and static force matrices. The program also creates a boundary condition index which will be used by succeeding programs to reduce the global dynamic force matrix, and to add the inertial effects of moving nodes into the global formulation (ie, to finish applying the essential boundary conditions).

The program will be executed if the chosen solution method is modal analysis or dynamic reduction of the structure. When the program completes its execution, it will check the specified solution-pathway and chain to the appropriate program. There are two pathways possible, the program will chain to DynFEP.eigen solver if modal analysis is the chosen solution method or it will

chain to DynFEP.reduce if dynamic reduction is the chosen solution method.

#### DynFEP.reduce

This program reads the reduce index created by DynFEP.create (from user input) and reduces the number of equations using the methods presented. In addition the program prepares two matrices (stored in temporary disk files) which are used by DynFEP.uncouple/solve to transform the primary eigen vectors (eigen vector of the reduced structure) into a eigen vector describing the full structure.

The program is executed along the dynamic reduction solution pathway. It executes after DynFEP.essential BC. When it completes execution it chains to the DynFEP.eigen solver program.

#### DynFEP.eigen solver

This program loads prepared mass and stiffness matrices and solves the corresponding eigen value problem using the Stodola method and a sweeping matrix as presented above. The result of the program are computed matrices of the eigenvalues and eigen vectors.

The program assumes that the global matrices have had the rows and columns of constrained degrees of freedom (ie, specified static and/or dynamic displacement) removed. The program uses the five variables passed to it by the DynFEP.menu program to determine if the global matrices have been reduced. If the matrices have been reduced, the program finds the transformation matrix for each mode frequency (using information prepared by DynFEP.reduce) and uses it to transform the partial eigen vectors into a full eigen vectors.

The DynFEP.eigen solver will execute if the dynamic reduction or modal analysis method of solution is chosen. The program chains to DynFEP.uncouple/solve upon completion.

#### DynFEP.uncouple/solve

This program pulls together the work of previous programs and solves the problem to its completion. Its execution results in a displacement -vs- time history, for each node, stored on disk. The displacements recorded are relative to the movement of the base. Information provided by the user regarding specified node displacement is assumed to be accelerations of uniform base movement.



The DynFEP.uncouple/solve program loads the static force matrix, the boundary conditions index, the mode shape matrix, the eigenvalues, and the mass matrix. If the structure has not been reduced the program also loads the initial conditions and converts them to the generalized coordinates to be used to start the Newmark direct integration scheme (If the structure is reduced the mode shape matrix is not square and can not be inverted to find the initial conditions generalized form. Therefore, if the structure is reduced, all initial conditions must equal zero). The program then finds the generalized mass matrix and starts the numerical integration scheme.

The first operation for each time step is to find the new dynamic force matrix including the inertial effects of base motion. The dynamic forces for applied to nodes or members are first found for every node in the structure; the dimension of the matrix in this form is  $3(GN) \times 1$  (where  $3(GN)$  means three times the number of global nodes). Then the essential boundary conditions are applied, resulting in a  $n \times 1$  matrix (where  $n$  is equal to the number of unknown node displacements). The inertia forces are obtained by multiplying a base acceleration matrix by the mass matrix (after essential boundary conditions, therefore matrix is  $n \times 1$ ). The dynamic, inertial, and static force matrices are then summed. Finally the force matrix is transformed to its generalized form by premultiplying it with the transpose of the mode shape matrix. The dimension of the generalized force matrix is either  $n \times 1$ , or  $m \times 1$  if the structure has been reduced (where  $m$  is equal to the number of retained modes).

The equations are now in their uncoupled form. The Newmark method is applied to solve for the generalized displacements, velocities, and accelerations for the current time step. Once the generalized displacements are calculated the real displacements are found by premultiplying the generalized displacements, and their derivatives, by the mode shape matrix. The real displacements are then stored and the program goes to the next time step. The program proceeds for a specified number of time steps.

The DynFEP.uncouple/solve program is the ending program for either the dynamic reduction solution method or modal analysis. Upon completion the program chains to the DynFEP.menu program.

### DynFEP

The DynFEP program loads the full mass and stiffness matrices, and the initial conditions. Using the Newmark method and Gauss elimination it solves the problem in its complete form. The result of the program is a displacement -vs- time history of every node in the system. The displacements recorded are absolute with regards to the coordinate system. Information provided by the user regarding movement of nodes is assumed to absolute displacement also.

The first operation for each time step is to find the new dynamic force matrix. The program reviews the node and element loading and displacement information stored in the structure data files (displacement information is assumed to be absolute displacement). If appropriate time history files for non-harmonic forces will also be accessed. The new dynamic force matrix is constructed and added to the static force matrix. During this process the program also constructs a boundary condition index.

The Newmark method is applied, then the boundary condition index is used to apply the essential boundary conditions. The essential boundary conditions are applied by modifying the equations which express the known displacement. The program will again access node information stored on disk to find the specified displacement, accessing time history files where appropriate for non-harmonic displacement of nodes.

The equations are now solved using a Gauss elimination technique, time is incremented and the process is repeated. The program continues for a specified number of time steps.

The DynFEP program executes only when this method of solution has been chosen. When the program completes execution, it returns control to the DynFEP.menu program.

### Data Files

Information describing the structure are contained in five primary files, the information file, the node file, and the element file, initial conditions, and reductions (when dynamic reduction is to be used). These files are created by DynFEP.create from information provided by the user. A summary of these data files and their structure is presented below.

Information Data File:

Name: <Structure Name>

Type: permanent, sequential text

Info:	Field	Field	
<u>Description</u>	<u>Length</u>	<u>Name</u>	<u>Remarks</u>
number of global nodes	n/a		
number of elements	n/a		
number unknown displacements	n/a		
number of retained modes	n/a		

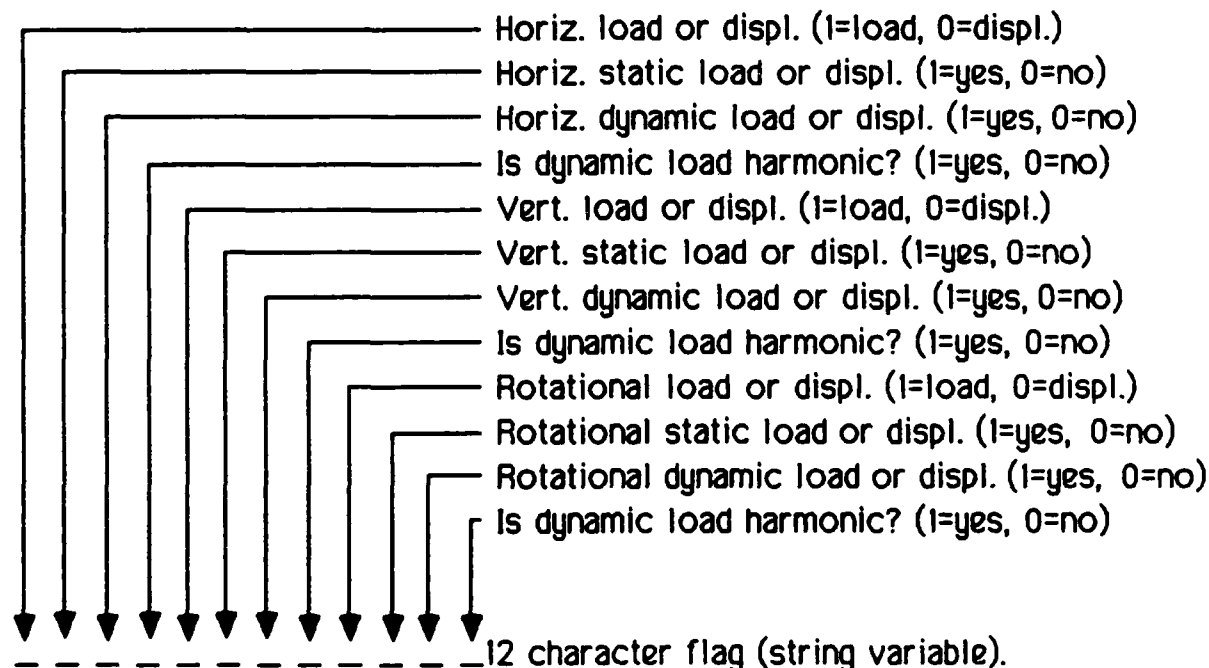
Node Data File:

Name: <Structure Name>.Nodes

Type: permanent, random access

Info:	Field	Field	
<u>Description</u>	<u>Length</u>	<u>Name</u>	<u>Remarks</u>
Flag describing boundary cond.	12	Flg1\$	string variable
X global coordinate	4	N\$(1)	single precision
Y global coordinate	4	N\$(2)	single precision
Horz. Static Load or Displ.	4	N\$(3)	single precision
Dyn. Load or Displ. Amp.	4	N\$(4)	single precision
dynamic frequency	4	N\$(5)	single precision
dynamic phase angle	4	N\$(6)	single precision
time history file name	8	N\$(7)	string variable
Vert. Static Load or Displ.	4	N\$(8)	single precision
Dyn. Load or Displ. Amp.	4	N\$(9)	single precision
dynamic frequency	4	N\$(10)	single precision
dynamic phase angle	4	N\$(11)	single precision
time history file name	8	N\$(12)	string variable
Rot. Static Load or Displ.	4	N\$(13)	single precision
Dyn. Load or Displ. Amp.	4	N\$(14)	single precision
dynamic frequency	4	N\$(15)	single precision
dynamic phase angle	4	N\$(16)	single precision
time history file name	8	N\$(17)	string variable

Meaning of flag variable:



The above file structure allows the user to specify both a static and a dynamic load or displacement at any node (the modal methods of solution do not support independent displacement of nodes). The DynFEP programs will interpret the stored information to be either a specified load or a specified displacement depending on the above twelve character flag. The flag also tells the program to whether or not to look for static or dynamic loads and whether the dynamic loads are harmonic or non-harmonic. The inclusion of a phase angle allows nodes to be loaded or displaced out of phase of one another for harmonic displacement or loading.

Element Data File:

Name: <Structure Name>.Elements

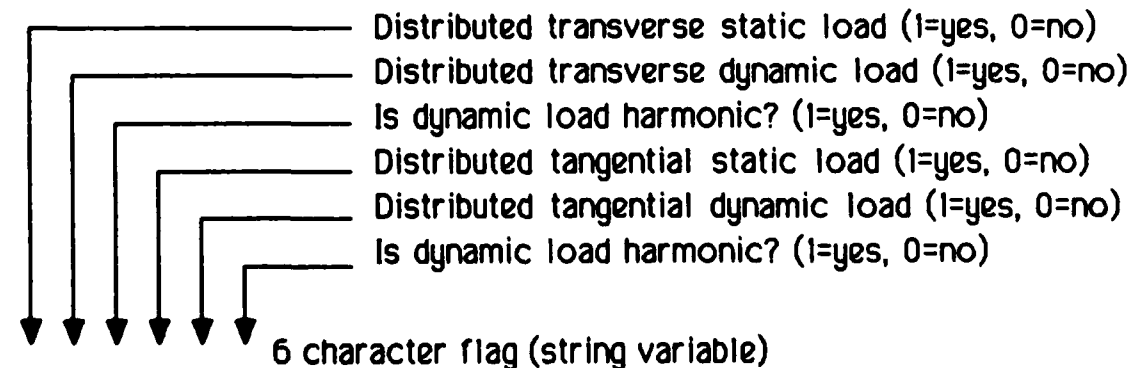
Type: randomaccess

Info:

<u>Description</u>	<u>Field Length</u>	<u>Field Name</u>	<u>Remarks</u>
Flag describing element loading	6	Flg2\$	string variable
Global node number of left side	2	Lt\$	integer
Global node number of right side	2	Rt\$	integer

<u>Description</u>	<u>Field Length</u>	<u>Field Name</u>	<u>Remarks</u>
Left side moment of inertia, $I_1$	4	E\$(1)	single precision
cross sectional area, $A_1$	4	E\$(2)	single precision
mass per unit length, $m_1$	4	E\$(3)	single precision
Right side moment of inertia, $I_2$	4	E\$(4)	single precision
cross sectional area, $A_2$	4	E\$(5)	single precision
mass per unit length, $m_2$	4	E\$(6)	single precision
Modulus of elasticity, E	4	E\$(7)	single precision
Transverse static load left side	4	E\$(8)	single precision
static load right side	4	E\$(9)	single precision
dynamic amplitude	4	E\$(10)	single precision
dynamic frequency	4	E\$(11)	single precision
dynamic phase angle	4	E\$(12)	single precision
name of time history file	4	E\$(13)	single precision
Tangential static load left side	4	E\$(14)	single precision
static load right side	4	E\$(15)	single precision
dynamic amplitude	4	E\$(16)	single precision
dynamic frequency	4	E\$(17)	single precision
dynamic phase angle	4	E\$(18)	single precision
name of time history file	4	E\$(19)	single precision

Meaning of flag variable:



The above file structure allows the user to apply static and dynamic loads at the same time. In addition static distributed loads can vary linearly (they can have

different values at each side of the element). Though use of the above flag, element loading may also be harmonic or non-harmonic.

Displacement History File:

Name: <Structure Name>.displ

Type: permanent, random access

Info:		Field	Field	
<u>Description</u>	<u>Length</u>	<u>Name</u>	<u>Remarks</u>	
displacement of node	8	n/a	( actual or relative, )	
velocity of node	8	n/a	( see program notes )	
acceleration of node	8	n/a	( for explanation. )	

The purpose of this file is to store the initial conditions of the structure, as defined by the user and to store the displacement -vs- time history of the structure after solution. Each record contains the displacement, velocity, and acceleration of the appropriate degree of freedom of the node. The first 3(GN) (3 degrees of freedom times the number of global nodes) records the initial conditions of the structure, t=0 (supplied by the user). The next 3(GN) records report the conditions of the structure at time step 1, and so on.

Reduction File:

Name: <Structure Name>.reduce

Type: permanent, random access

Info:		Field	Field	
<u>Description</u>	<u>Length</u>	<u>Name</u>	<u>Remarks</u>	
1 <sup>st</sup> reduction of Node/DOF	8	n/a	* between 1 and 3(GN)	
2 <sup>nd</sup> reduction of Node/DOF	8	n/a	* between 1 and 3(GN)	
.....	8	n/a	* between 1 and 3(GN)	
i <sup>th</sup> reduction of Node/DOF	8	n/a	* between 1 and 3(GN)	

The purpose of the above file is to store a list of equations to retain in the matrix formulation. The DynFEP.create data file constructs the above file with information provided by the user.

To support the use of non-harmonic forcing functions or specified displacements of nodes, the programs are capable of reading a time history file. Each record of the file contains a time and a magnitude of the force or displacement. The programs will interpolate between two time steps if the required time is not on

file. A rapid search is employed by the program to find appropriate time, it assumes that the file is sequential in time. The first record in the file must contain the total number of time steps recorded in the history file.

It should also be noted that the DynFEP program assumes that information presented in this file is absolute displacement, while the DynFEP.uncouple/solve program assumes that the information is accelerations.

User defined Force or Displacement History File:

Name: <User specified file name>

Type: permanent, random access

Info:		Field	Field	
<u>Description</u>	<u>Length</u>	<u>Name</u>	<u>Remarks</u>	
Number of time steps in file	8	n/a	First record only	
Not used	8	n/a	First record only	
Time	8	n/a	normal record	
Displacement or Force magnitude	8	n/a		

During the solution of the problem other permanent and temporary files will be created. The purpose of these data files is first to provide storage of matrices necessary in the solution and there by reduce the amount of memory space required. Secondly, these data files eliminate the need to recalculate matrices to analyze different loading conditions or use alternative solutions. A summary of these data files is presented below.

Permanent data files:

<u>File name</u>	<u>Description</u>	<u>Size</u>
<Structure Name>.K&F.c	stiffness and static forces before BC	$3(GN) \times 3(GN) + 1$
<Structure Name>.M.c	mass matrix before essential BC	$3(GN) \times 3(GN)$
<Structure Name>.K&F	stiffness and static forces after BC	$n \times (n+1)$
<Structure Name>.M	mass matrix after essential BC	$n \times n$
<Structure Name>.K*	reduced stiffness and static forces	$m \times (m+1)$
<Structure Name>.M*	reduced mass matrix	$m \times m$
<Structure Name>.reduce	listing of nodes to be retained	$m \times 1$
<Structure Name>.S	structure mode shapes	$m \times m$
<Structure Name>.eigen	eigenvalues of structure	$n \times m$

Temporary data files:

<u>File name</u>	<u>Description</u>	<u>Size</u>
<Structure Name>.Kpp	partitioned stiffness matrix	$m \times m$
<Structure Name>.Kps	partitioned stiffness matrix	$m \times (n-m)$
<Structure Name>.Ksp	partitioned stiffness matrix	$(n-m) \times m$
<Structure Name>.Kss	partitioned stiffness matrix	$(n-m) \times (n-m)$
<Structure Name>.P1	needed to calc mode shape [T]	$(n-m) \times m$
<Structure Name>.P2	needed to calc mode shape [T]	$(n-m) \times m$
<Structure Name>.D	dynamic matrix $[K]^{-1} [M]$	$m \times m$

Where  $n$  is equal to the number of unknown displacements and  $m$  is equal to the number of modes retained in the answer. Note that if the structure is not reduced then  $m$  is equal to  $n$ .



## Conclusions

A set of micro-computer programs capable of analyzing the dynamic behavior plane frame structures has been developed from the set of assumed governing differential equations. The system of programs allow different approaches to be used in analyzing a dynamic structure. The capability to use different approaches provide a means of building confidence by comparing the results of the different methods, and the flexibilities provided by the different approaches.

Each solution approach presented has unique abilities and limitations. While the straight numerical integration performed by DynFEP has the ability to include the independent displacement of nodes, it must process the formulation with no reduction or further simplification. The added capability has come at the cost of not being able to analyze larger structures and in a longer computation time. The modal analysis approach presented offers a faster computation time but sacrifices the ability to effectively handle independent movement of nodes. The dynamic reduction approach offers the ability to do larger structures with little or no additional computation time, but at some sacrifice for accuracy.

The choice of which solution approach to use will normally be based on the type of problem to be solved. As an example, in Civil Engineering programs such as these would be used primarily for the analysis of structure response to earthquakes. The dynamic reduction approach presented above would be most useful in this situation as it offers the best computation time to size advantage and can support the boundary conditions imposed by an earthquake.

The programs as presented here are capable of handling about 50 nodes when run on a system with 370K of memory available (assuming 8 bytes of memory is required for each double precision variable used). There are two primary approaches which could be employed to increase the capacity of the programs. First the assemblage of the global matrices could be done in upper-banded form. Since mass and stiffness matrices are normally very sparse, this would substantially increase the capacity. Secondly the global mass and stiffness matrices could be done on disk rather than in core (also the application of boundary conditions, and the reduction of the mass and stiffness matrices).

While this would slow down execution because of increased I/O activity, the capability to handle larger structures is limited only by the amount of disk space and the number of retained modes. Since a typical micro-computer system in a professional installation will include between 10 and 20 megabytes of hard disk storage, the analysis of structures with several hundred nodes is seen to be reasonable.

In addition to increasing the capacity of the programs presented, there are other enhancements which would make them more valuable as an engineering tool. The programs now simply grind through a specified number of time steps. It would be desirable for them to have the ability to check selected parameters during the processing and determine for themselves when to stop the processing. Such parameters might include critical stresses in selected members, a maximum stress in any member, completion of a full cycle of all forcing functions, achievement of a maximum deflection, etc.

The ability to handle three dimensional plane frames is a very natural expansion to the program capabilities. Other enhancements could include an interactive input of information in a CAD/graphics orientated format, and graphic replay of structure response.

All of these enhancements are within the current computing capability of today's micro-computers. While the analysis on truly complex structures will remain in the domain of mainframe computers, the dynamic analysis of small structures is within the realm of processing by micro-computer systems. Such smaller structures are those that can be described in several hundred nodes or less, or simplifications of more complex structures which are being used for preliminary design or investigation prior to a more detailed analysis on a mainframe. The methods and programs presented in this paper form a corner stone for building the enhanced systems which are required to fill this expanding field of engineering analysis.

## Bibliography

Guyan, R. J., *"Reduction of Stiffness and Mass Matrices,"* American Institute of Aeronautics and Astronautics Journal, Vol. 3, No. 2, Feb., 1965.

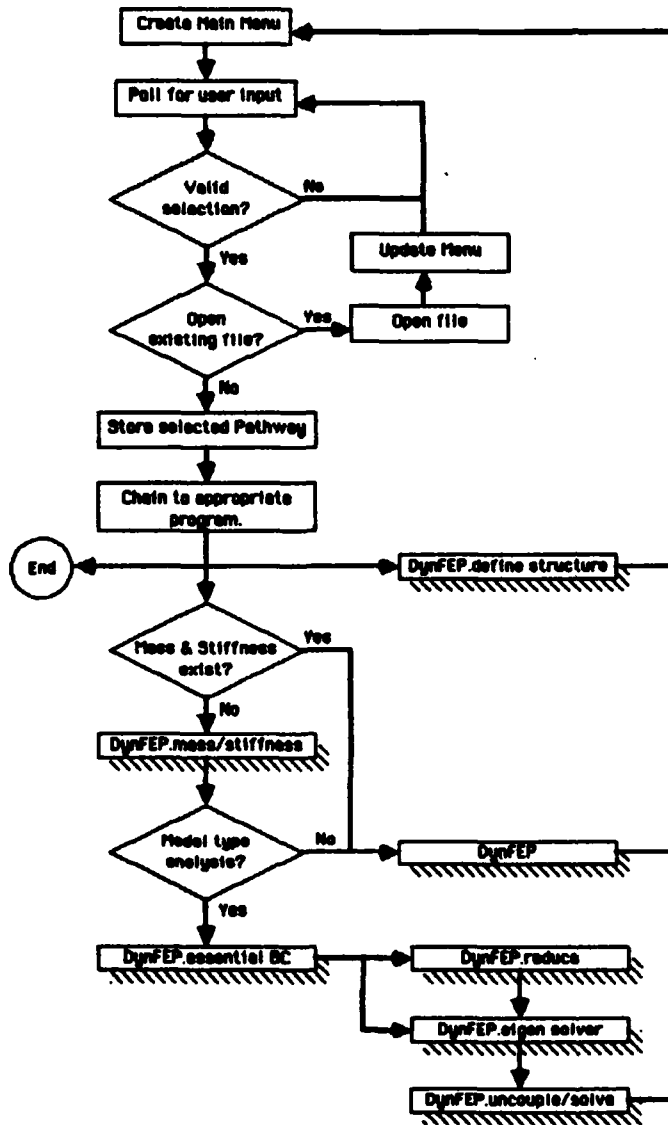
Clough, R. W. and Penzien, J., *"Dynamics of Structures"*. McGraw-Hill Book company: New York, New York, 1975.

Bathe, Klaus-Jurgen, and Wilson, E. L. , *"Numerical Methods in Finite Element Analysis"*. Prentice-Hall, Inc: Englewood Cliffs, New Jersey, 1976

Miller, C. A., *"Dynamic Reduction of Structural Models"*. Journal of the Structural Division, Proceedings of the American Society of Civil Engineers, Vol.106 No.ST10, Oct., 1980.

Reddy, J. N., *"An Introduction to the Finite Element Method"*. McGraw-Hill Book company: New York, New York, 1984.

# DynFEP.menu Flow Diagram



## Notes:

### Program Variables:

GN = number of global nodes.  
 NE = number of elements.  
 n = number of unknown displacements in structure.  
 m = number of retained modes (if no reductions then  $m=n$ ).  
 Path\$ = string defining the chosen method of solution.

### Valid Menu Selections:

If no structure data file is open, then valid selections include, Open File, and Create New File.  
 If a structure data file is opened, then all selections are valid.

```

/ +-----+
/ |      DynFEP.menu      |
/ +-----+

```

COMMON GN,NE,DOF,n,m,PN\$,Path\$

Start: 'set up menu

```

WINDOW 1,,(110,30)-(365,54),2: GOSUB BigText
WINDOW 1:CALL MOVETO(20,16): PRINT "Dynamic Finite Element Program";
WINDOW 2,,(130,70)-(350,306),4
BUTTON 1,1,"Create New Data File",(20,20)-(200,40),1
BUTTON 2,1,"Open Existing Data File",(20,45)-(200,65),1
BUTTON 3,0,"Direct Integration Only",(20,80)-(200,100),1
BUTTON 4,0,"Modal Analysis",(20,105)-(200,125),1
BUTTON 5,0,"Dynamic Reduction",(20,130)-(200,150),1
BUTTON 6,1,"Quit",(120,165)-(200,185),1
IF LEN(PN$)>0 THEN GOSUB Look.at.File
GOSUB NormalText

```

loop:

```

CALL MOVETO(10,200): PRINT "Problem name = ";PN$
CALL MOVETO(10,212): PRINT USING "Global Nodes =##";GN
CALL MOVETO(10,224): PRINT USING "Elements =##";NE
WHILE DIALOG(0)<1: WEND 'wait for the user to do something
ON DIALOG(1) GOTO Create,Existing,FEM,Modal,Reduce,Quit 'branch according to menu selection

```

```

Create: Path$="Create New Data File": GOSUB Change.Window
F$="Basic Disk 1:DynFEP.create data file"
CHAIN F$: END

```

Existing:

```

PN$=FILES$(1,"DYNA") ' get Problem Name from user
IF PN$<>"" THEN GOSUB Look.at.File
GOTO loop

```

FEM: Path\$="Direct Integration Only": GOSUB Change.Window

```

IF n<0 THEN F$="Basic Disk 1:DynFEP.mass/stiffness" ELSE F$="Basic Disk 1:DynFEP" 'no need to reassemble gobal matric
es.
CHAIN F$: END

```

Modal: Path\$="Modal Analysis": GOSUB Change.Window

```

IF n<0 THEN F$="Basic Disk 1:DynFEP.mass/stiffness" ELSE F$="Basic Disk 1:DynFEP.essential BC" 'no need to reassemble
gobal matrices.
CHAIN F$: END

```

Reduce: Path\$="Dynamic Reduction": GOSUB Change.Window

```

IF n<0 THEN F$="Basic Disk 1:DynFEP.mass/stiffness" ELSE F$="Basic Disk 1:DynFEP.essential BC" 'no need to reassemble
gobal matrices.
CHAIN F$: END

```

```

Quit: Path$="Output Window": GOSUB Change.Window
END

```

-----  
Subroutines Below  
-----

CE-685  
Term Project

Larry Goshorn  
August 1985

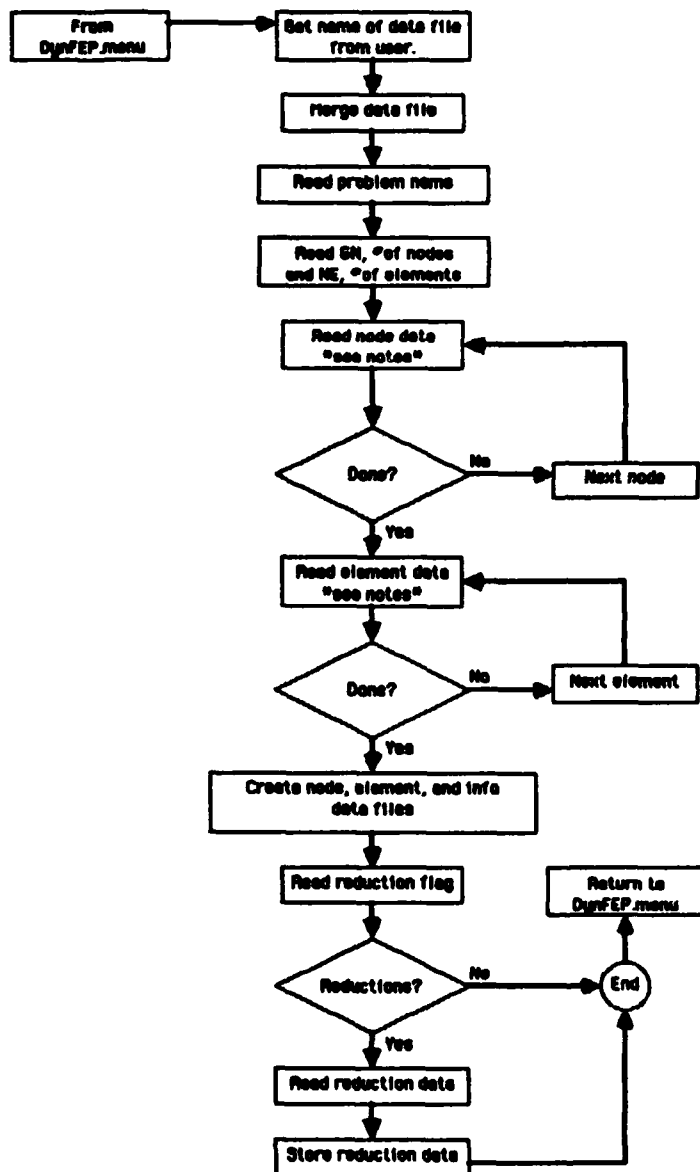
Look.At.File: DOF=3        ' find status of processing  
  OPEN Pn% FOR INPUT AS #1: INPUT#1,GN,NE,n,m: CLOSE #1  
  FOR i=3 TO 5: BUTTON i,1: NEXT i 'activate buttons  
RETURN

Change.Window: WINDOW CLOSE 1: WINDOW CLOSE 2: CLOSE  
  WINDOW 1,Path\$(5,40)-(265,298),1: RETURN

BigText:CALL TEXTFONT(0):CALL TEXTSIZE(12):RETURN ' Chicago  
LittleText:CALL TEXTFONT(1):CALL TEXTSIZE(9):RETURN ' Geneva  
NormalText:CALL TEXTFONT(1):CALL TEXTSIZE(10):RETURN ' Geneva  
FormattedText:CALL TEXTFONT(4):CALL TEXTSIZE(9):RETURN ' Monaco

# DynFEP.create data file

Flow Diagram



## Notes:

### Program Variables:

GN = number of global nodes.  
 NE = number of elements.  
 n = number of unknown displacementss in structure.  
 m = number of retained nodes (if no reduction then  $m=n$ )  
 Path\$ = string variable indicating the choosen method of solution.

The program assumes that a data Text file has been prepared. Format of the file is that of a BASIC DATA statement. All data shown in the discription of the data file structure must be included.

```
' +-----+
' | DynFEP.create data file |
' +-----+
```

COMMON GN,NE,DOF,n,m,PN\$,Path\$

```
WINDOW 2,,(110,250)-(380,300),2: GOSUB BigText: WINDOW 2
PRINT "If you have not created a text file of"
PRINT "Basic DATA statements for this program,"
PRINT "press the 'Cancel' button!"CHR$(7);
PN$=FILES$(1,"TEXT") 'get data file name from user
WINDOW CLOSE 2: IF PN$="" THEN STOP 'user hit cancel button
PRINT "Type 'GOTO Start' to continue."CHR$(7)
MERGE PN$ 'load user created data file & execute with it
```

Start: DIM N\$(17),E\$(19):Z/=-1

GOSUB FormatedText

READ PN\$

F\$=PN\$+".nodes": OPEN F\$ AS #1 LEN=92

FIELD#1,12 AS Flg1\$, 4 AS N\$(1), 4 AS N\$(2), 4 AS N\$(3), 4 AS N\$(4), 4 AS N\$(5), 4 AS N\$(6), 8 AS N\$(7), 4 AS N\$(8), 4 AS N\$(9), 4 AS N\$(10), 4 AS N\$(11), 8 AS N\$(12), 4 AS N\$(13), 4 AS N\$(14), 4 AS N\$(15), 4 AS N\$(16), 8 AS N\$(17)

F\$=PN\$+".elements": OPEN F\$ AS #2 LEN=94

FIELD#2,6 AS Flg2\$,2 AS Lt\$,2 AS Rt\$,4 AS E\$(1),4 AS E\$(2),4 AS E\$(3),4 AS E\$(4),4 AS E\$(5),4 AS E\$(6),4 AS E\$(7),4 AS E\$(8),4 AS E\$(9),4 AS E\$(10),4 AS E\$(11),4 AS E\$(12),8 AS E\$(13),4 AS E\$(14),4 AS E\$(15),4 AS E\$(16),4 AS E\$(17),4 AS E\$(18),8 AS E\$(19)

F\$=PN\$: OPEN F\$ FOR OUTPUT AS #5

READ NumNodes%

FOR i=1 TO NumNodes%

READ a\$: LSET Flg1\$=a\$

FOR j=1 TO 17

IF j=7 OR j=12 OR j=17 THEN READ a\$: LSET N\$(j)=a\$ ELSE READ a: LSET N\$(j)=MK\$(a)

NEXT j

PUT#1,i

NEXT i

READ NumElements%

FOR i=1 TO NumElements%

READ a\$,Lt%,Rt%: LSET Flg2\$=a\$: LSET Lt\$=MK\$(Lt%): LSET Rt\$=MK\$(Rt%)

FOR j=1 TO 19

IF j=13 OR j=19 THEN READ a\$: LSET E\$(j)=a\$ ELSE READ a: LSET E\$(j)=MK\$(a)

NEXT j

PUT#2,i

NEXT i

'\*\*\*\*\* debug

'TRON

'\*\*\*\*\*

DIM UN(NumNodes%\*3,3) ' initial conditions

FOR i=1 TO NumNodes%\*3

READ UN(i,1),UN(i,2),UN(i,3)

NEXT i: n3=3: n=NumNodes%\*3

CALL Display.Matrix(n,n3,UN(),"Initial Conditions")

CALL store.Matrix(n,n3,UN(),PN\$+".initial",n3)

'\*\*\*\*\* debug



CE-685  
Term Project

Larry Goshorn  
August 1985

'TROFF

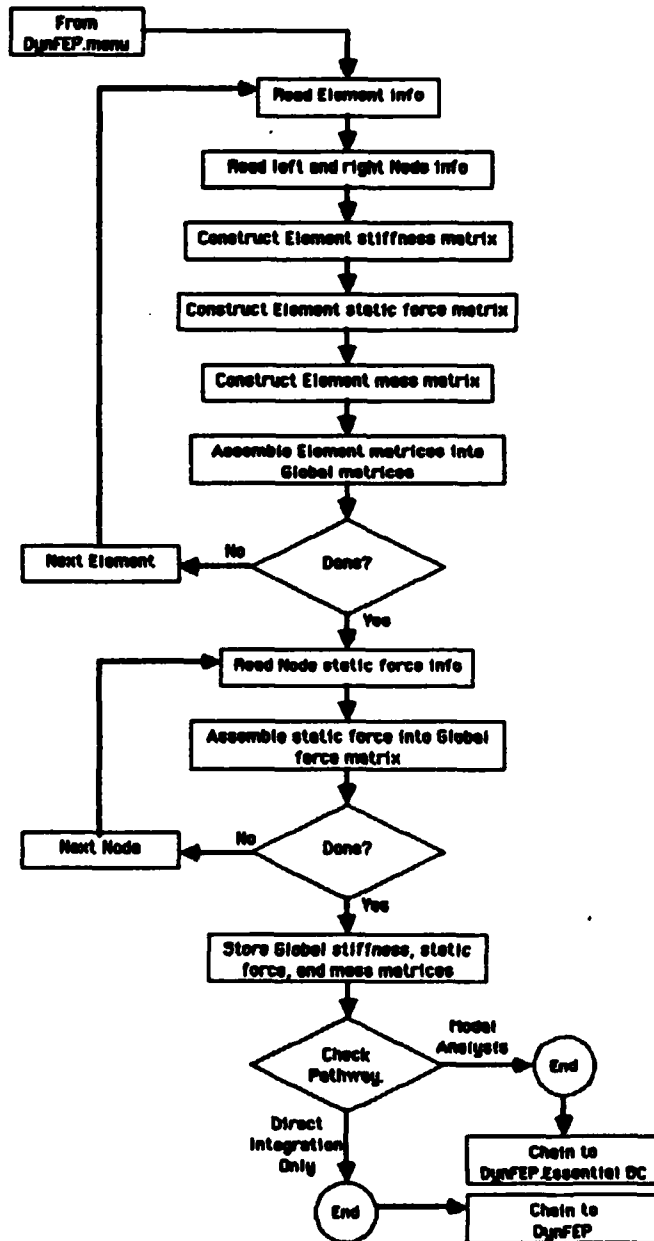
'\*\*\*\*\*

```
INPUT "Reduction info in this data (y/n)";a$
n=0: IF a$(">y") THEN GOTO finish.up
READ m: OPEN PNs+".reduce" AS #4 LEN=8: FIELD#4,8 AS aa$
FOR i=1 TO m: READ r: LSET aa$=MKD$(r): PUT#4,i: NEXT i
```

finish.up:

```
INPUT "Have global matrices been assembled (y/n)";a$
IF a$(">y") THEN n=-1 ELSE n=0
WRITE#5,NumNodes%,NumElements%,n,n
GN=NumNodes%: NE=NumElements%: DOF=3
CLOSE: NAME PNs AS PNs,"DYNA"
```

# DynFEP.mass/stiffness Flow Diagram



## Notes:

### Program Variables:

GN = number of global nodes.

NE = number of elements.

n = number of unknowns in the structure.

m = number of retained modes (if no reduction then  $m=n$ ).

Path\$ = string variable indicating the chosen method of solution.

[K] provides storage for the global stiffness matrix, and the static force matrix.

The stiffness matrix is a square matrix with dimensions equal to GN times 3.

The static force matrix is stored with the stiffness in an addition column.

[M] provides storage for the global mass matrix. It is a square matrix with dimensions equal to GN times 3.

```

/ -----
/ | DynFEP.mass/stiffness |
/ -----

```

COMMON GN,NE,DOF,n,m,PV\$,Path\$: GOTO Start

Subroutines below

AssembleForceMat:

```

FOR i=1 TO GN: GET#1,i ' read specified nodal loads
FOR j=0 TO DOF-1
index=(i-1)*DOF+j+1:k=3+5*j
'---NodeStatForces:
Sload=0: A=1+j*4
IF MID$(F1g1$,A,2)="11" THEN Sload=CVS(N$(k)) ' static load
KN(index,n+1)=KN(index,n+1)+Sload ' add in force; positive to right, upward, clockwise
'---
NEXT j,i: RETURN

```

ElementMatrixAssembler:

```

FOR ELEMENT=1 TO NE
GOSUB BuildElementMatrices
IR=(N1%-1)*DOF:IC=(N2%-1)*DOF
'---Assem.K.M.Stat.Elem.Forces:
FOR i=1 TO DOF
KN(IR+i,n+1)=KN(IR+i,n+1)+AN(i,7) ' element forces stored in column Nn+1
KN(IC+i,n+1)=KN(IC+i,n+1)+AN(i+DOF,7)
FOR j=1 TO DOF
KN(IR+i,IR+j)=KN(IR+i,IR+j)+AN(i,j) ' assemble stiffness matrix
KN(IR+i,IC+j)=KN(IR+i,IC+j)+AN(i,j+DOF)
KN(IC+i,IR+j)=KN(IC+i,IR+j)+AN(i+DOF,j)
KN(IC+i,IC+j)=KN(IC+i,IC+j)+AN(i+DOF,j+DOF)
MN(IR+i,IR+j)=MN(IR+i,IR+j)+BN(i,j) ' assemble mass matrix
MN(IR+i,IC+j)=MN(IR+i,IC+j)+BN(i,j+DOF)
MN(IC+i,IR+j)=MN(IC+i,IR+j)+BN(i+DOF,j)
MN(IC+i,IC+j)=MN(IC+i,IC+j)+BN(i+DOF,j+DOF)
NEXT j,i
'---
NEXT element

```

BuildElementMatrices:

```

GET#2,ELEMENT:N1%=CVI(Lt$):N2%=CVI(Rt$) ' get left and right global node #'s
GET#1,N1%:X1=CVS(N$(1)):Y1=CVS(N$(2)) ' get left side coord's
GET#1,N2%:X2=CVS(N$(1)):Y2=CVS(N$(2)) ' get right side coord's
L=SQR((Y1-Y2)^2+(X1-X2)^2) ' find element length
GOSUB Elem.K.M.Stat.Forces
IF X1-X2=0 THEN angle=SGN(Y1-Y2)*Pi#2 ELSE angle=2*Pi#-ATN((Y1-Y2)/(X1-X2))
IF angle>2*Pi#+.003 OR angle<2*Pi#-.003 THEN GOSUB Transform
RETURN

```

Elem.K.M.Stat.Forces:

```

I1=CVS(E$(1)):I2=CVS(E$(4)) ' moments of inertia
A1=CVS(E$(2)):A2=CVS(E$(5)) ' areas
m1=CVS(E$(3)):m2=CVS(E$(6)) ' mass/length
E=CVS(E$(7)) ' elastic modulus

```

```

FOR i=1 TO 6:FOR j=1 TO 7:AN(i,j)=0:NEXT j,i ' initialize/build element stiffnesses
AN(1,1)=E*(A1+A2)/(2*L):AN(4,4)=AN(1,1):AN(1,4)=-AN(1,1)
AN(2,2)=E*(I1+I2)/L^3:AN(5,5)=AN(2,2):AN(2,5)=-AN(2,2)
AN(2,3)=-E*(4*I1+2*I2)/L^2:AN(3,5)=-AN(2,3)
AN(2,6)=-E*(2*I1+4*I2)/L^2:AN(5,6)=-AN(2,6)
AN(3,3)=E*(3*I1+I2)/L
AN(3,6)=E*(I1+I2)/L
AN(6,6)=E*(I1+3*I2)/L
FOR i=2 TO 6:FOR j=1 TO i-1:AN(i,j)=AN(j,i):NEXT j,i ' symetrize stiffness
FOR i=1 TO 6:FOR j=1 TO 6:BN(i,j)=0:NEXT j,i ' initialize/build element matrix
BN(1,1)=70*L*(3*m1+m2)/840
BN(1,4)=70*L*(m1+m2)/840
BN(2,2)=24*L*(10*m1+3*m2)/840
BN(2,3)=-2*L^2*(15*m1+7*m2)/840
BN(2,5)=54*L*(m1+m2)/840
BN(2,6)=2*L^2*(7*m1+6*m2)/840
BN(3,3)=L^3*(5*m1+3*m2)/840
BN(3,5)=-2*L^2*(6*m1+7*m2)/840
BN(3,6)=-3*L^3*(m1+m2)/840
BN(4,4)=70*L*(m1+3*m2)/840
BN(5,5)=24*L*(3*m1+10*m2)/840
BN(5,6)=2*L^2*(7*m1+15*m2)/840
BN(6,6)=L^3*(3*m1+5*m2)/840
FOR i=2 TO 6:FOR j=1 TO i-1:BN(i,j)=BN(j,i):NEXT j,i ' symetrize mass matrix
DSload1=0:DSload2=0:TSload1=0:TSload2=0 ' find element loading
IF MID$(Flg2$,1,1)="1" THEN DSload1=CVS(E$(8)):DSload2=CVS(E$(9)) ' distributed static load
IF MID$(Flg2$,4,1)="1" THEN TSload1=CVS(E$(14)):TSload2=CVS(E$(15)) ' tangential static load
AN(1,7)=L*(20*TSload1+10*TSload2)/60 ' positive to the right
AN(2,7)=L*(-15*DSload1+45*DSload2)/60 ' positive upward
AN(3,7)=-L^2*(3*DSload1+2*DSload2)/60 ' positive clockwise
AN(4,7)=L*(10*TSload1+20*TSload2)/60 ' positive to the right
AN(5,7)=L*(9*DSload1+21*DSload2)/60 ' positive upward
AN(6,7)=L^2*(2*DSload1+3*DSload2)/60 ' positive clockwise
RETURN

```

Transform: 'Subroutine to transform Stiffness, Mass, and Force element matrices

```

GOSUB BuildTransformationMat
FOR i=1 TO 6:FOR j=1 TO 6:CN(i,j)=0:FOR k=1 TO 6:CN(i,j)=CN(i,j)+TN(k,i)*AN(k,j):NEXT k,j,i ' transpose[T]*[Ke]
FOR i=1 TO 6:FOR j=1 TO 6:AN(i,j)=0:FOR k=1 TO 6:AN(i,j)=AN(i,j)+CN(i,k)*TN(k,j):NEXT k,j,i '[transpose[T]*[Ke]]*[T]
FOR i=1 TO 6:CN(i,1)=0:FOR k=1 TO 6:CN(i,1)=CN(i,1)+TN(k,i)*AN(k,7):NEXT k,i:FOR i=1 TO 6:AN(i,7)=CN(i,1):NEXT 'tran[T]
*(Fe)
FOR i=1 TO 6:FOR j=1 TO 6:CN(i,j)=0:FOR k=1 TO 6:CN(i,j)=CN(i,j)+TN(k,i)*BN(k,j):NEXT k,j,i ' transpose[T]*[Me]
FOR i=1 TO 6:FOR j=1 TO 6:BN(i,j)=0:FOR k=1 TO 6:BN(i,j)=BN(i,j)+CN(i,k)*TN(k,j):NEXT k,j,i '[transpose[T]*[Me]]*[T]
RETURN

```

BuildTransformationMat: ' build [T]

```

FOR i=1 TO 6:FOR j=1 TO 6:TN(i,j)=0:NEXT j,i ' initialize
IF angle MOD Pi#2 THEN TN(1,1)=COS(angle) ELSE TN(1,1)=0
TN(4,4)=TN(1,1):TN(2,2)=TN(1,1):TN(5,5)=TN(1,1)
IF angle MOD Pi# THEN TN(1,2)=-SIN(angle) ELSE TN(1,2)=0
TN(4,5)=TN(1,2):TN(2,1)=-TN(1,2):TN(5,4)=TN(2,1)
TN(3,3)=1:TN(6,6)=1
RETURN

```

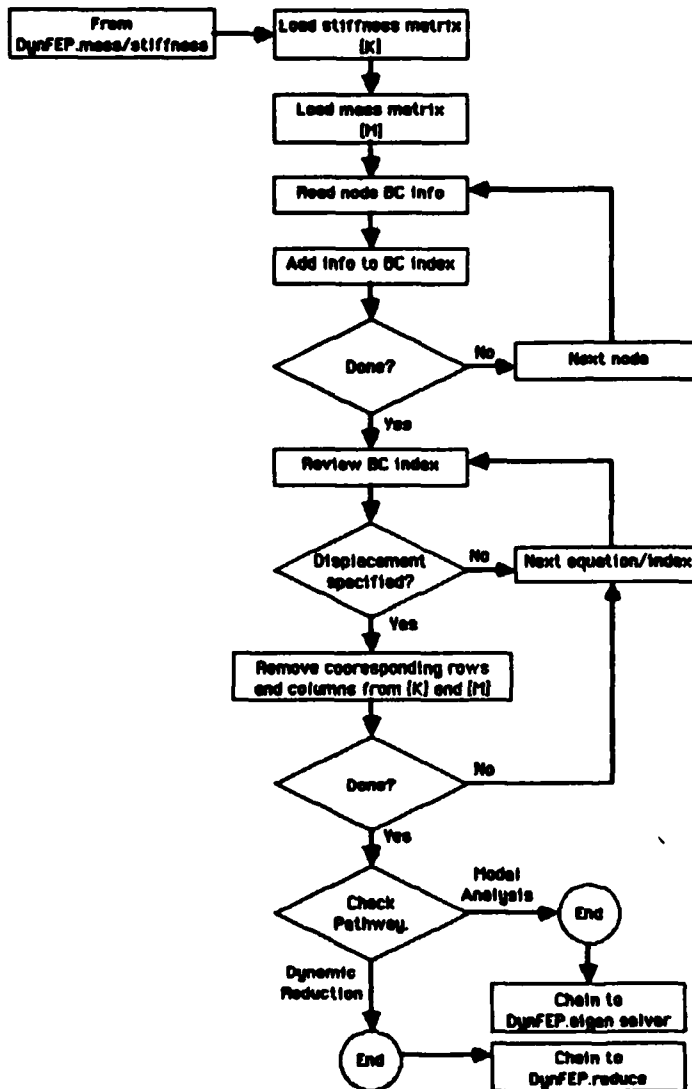
```
Start: n=GN*DOF: a1=1: a3=3: CR%=CHR$(13): Pi%=4*ATN(1)
DIM K$(n,n+1),M$(n,n),A$(6,7),B$(6,6),C$(6,6),T$(6,6),N$(17),E$(19)
F$=PN$+".nodes":OPEN F$ AS #1 LEN=92
FIELD#1,12 AS Flg1$,4 AS N$(1),4 AS N$(2),4 AS N$(3),4 AS N$(4),4 AS N$(5),4 AS N$(6),8 AS N$(7),4 AS N$(8),4 AS N$(9),4
AS N$(10),4 AS N$(11),8 AS N$(12),4 AS N$(13),4 AS N$(14),4 AS N$(15),4 AS N$(16),8 AS N$(17)
F$=PN$+".elements":OPEN F$ AS #2 LEN=94
FIELD#2,6 AS Flg2$,2 AS Lt$,2 AS Rt$,4 AS E$(1),4 AS E$(2),4 AS E$(3),4 AS E$(4),4 AS E$(5),4 AS E$(6),4 AS E$(7),4 AS E
$(8),4 AS E$(9),4 AS E$(10),4 AS E$(11),4 AS E$(12),8 AS E$(13),4 AS E$(14),4 AS E$(15),4 AS E$(16),4 AS E$(17),4 AS E$(
18),8 AS E$(19)
```

```
Build.Global.Matrices:
GOSUB ElementMatrixAssembler
ERASE A$,B$,C$,T$
```

```
Build.Static.Force.Mat:
DIM UO$(n,3),UI$(n,3),Q$(n)
GOSUB AssembleForceMat
CALL Store.Matrix(n,n+1,K$(),PN$+".K&F.c",a3) ' store stiffness and force matrices
CALL Store.Matrix(n,n,M$(),PN$+".M.c",a3) ' store stiffness and force matrices
'***** debug
CALL Display.Matrix(n,n+1,K$(),"Stiffness")
CALL Display.Matrix(n,n,M$(),"Mass")
'*****
n=0: OPEN PN$ FOR OUTPUT AS #3: WRITEN#3,GN,NE,n,m: CLOSE#3: NAME PN$ AS PN$,"DYNA"
CLOSE
IF Path$="Direct Integration Only" THEN CHAIN "Basic Disk 1:DynFEP" ELSE CHAIN "Basic Disk 1:DynFEP.essential BC"
END
```

-----  
SUB-Programs below  
-----

# **DynFEP.essential BC** Flow Diagram



## **Notes:**

### **Program Variables:**

- GN = number of global nodes.
- NE = number of elements.
- $n$  = number of unknowns in the structure.
- $m$  = number of retained modes (if no reduction then  $m=n$ ).
- Path\$ = string variable indicating the chosen method of solution.
- [K] provides storage for the global stiffness matrix, and the static force matrix. The stiffness matrix is a square matrix with dimensions equal to GN times 3. The static force matrix is stored with the stiffness in an addition column.
- [M] provides storage for the global mass matrix. It is a square matrix with dimensions equal to GN times 3.
- (BC) provides storage for the boundary condition index, a column of 1/0's. If 0 then displacement has been specified.

```

' +-----+
' | DynFEP.essential BC |
' +-----+

```

COMMON GN,NE,DOF,n,m,PN\$,Path\$

-----  
Subroutines Below  
-----

Switch:

```

FOR j=1 TO n: SWAP KN(i,j),KN(k,j): SWAP MN(i,j),MN(k,j): NEXT j
FOR j=1 TO n: SWAP KN(j,i),KN(j,k): SWAP MN(j,i),MN(j,k): NEXT j
RETURN

```

Start: n=GN\*DOF: a1=1: a2=2

DIM BC\$(n,1),KN(n,n+1),MN(n,n),N\$(17)

F\$=PN\$+".nodes":OPEN F\$ AS #1 LEN=92

FIELD#1,12 AS Flg1\$, 4 AS N\$(1), 4 AS N\$(2), 4 AS N\$(3), 4 AS N\$(4), 4 AS N\$(5), 4 AS N\$(6), 8 AS N\$(7), 4 AS N\$(8), 4 AS N\$(9), 4 AS N\$(10), 4 AS N\$(11), 8 AS N\$(12), 4 AS N\$(13), 4 AS N\$(14), 4 AS N\$(15), 4 AS N\$(16), 8 AS N\$(17)

CALL Retrieve.Matrix(n,n+1,KN(),PN\$+".K&F.c",a2)

CALL Retrieve.Matrix(n,n,MN(),PN\$+".M.c",a2)

Define.Essential.BC:

```

FOR i=1 TO GN: GET#1,i
FOR j=1 TO DOF
  index=(i-1)*DOF+j: k=1+4*(j-1)
  BC$(index,1)=VAL(MID$(Flg1$,k,1))
NEXT j,i

```

```

'***** debug
CALL Display.Matrix(n,a1,BC$(),"B. C.")
CALL Display.Matrix(n,n+1,KN(),"Stiffness")
CALL Display.Matrix(n,n,MN(),"Mass")
'*****
CALL Store.Matrix(n,a1,BC$(),PN$+".BC",a2)

```

Apply.Essential.BC: k=0

```

FOR i=1 TO n
  IF BC$(i,1)=1 THEN k=k+1: IF i(>)k THEN GOSUB Switch
NEXT i
k=0: FOR i=1 TO n: k=k+BC$(i,1): NEXT i: n=k
FOR i=1 TO n: SWAP KN(i,n+1),KN(i,GN*DOF+1): NEXT i
'***** debug
CALL Display.Matrix(n,n+1,KN(),"Stiffness")
CALL Display.Matrix(n,n,MN(),"Mass")
'*****
CALL Store.Matrix(n,n+1,KN(),PN$+".K&F",a2)
CALL Store.Matrix(n,n,MN(),PN$+".M",a2)
IF Path$="Modal Analysis" THEN m=n 'if not m has been set by DynFEP.create
OPEN PN$ FOR OUTPUT AS #2: WRITE#2,GN,NE,n,m: CLOSE#2: NAME PN$ AS PN$+".DYNA": CLOSE
IF Path$="Modal Analysis" THEN CHAIN "Basic Disk 1:DynFEP.eigen solver" ELSE CHAIN "Basic Disk 1:DynFEP.reduce"
END

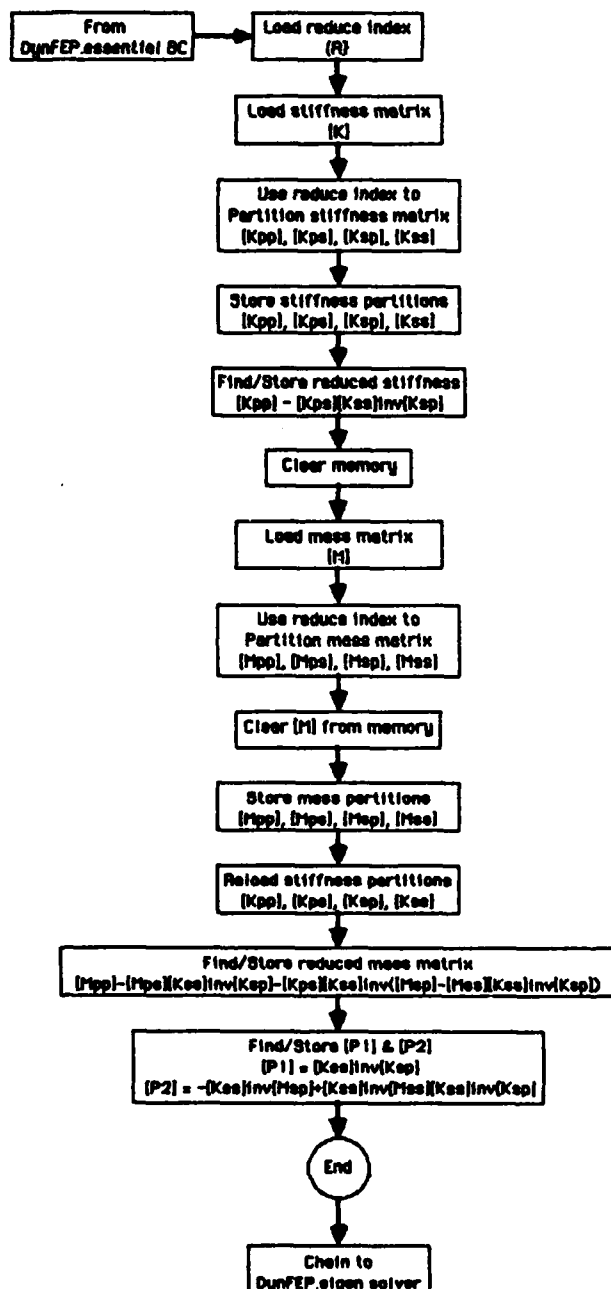
```

-----  
Sub-Programs Below  
-----

-----SUB Retrieve.Matrix(n,c,AN(),F\$,K) 37-700

## DynFEP.reduce

Flow Diagram



### Notes:

#### Program Variables:

GN = number of global nodes.  
 NE = number of elements.  
 n = number of unknowns in the structure.  
 m = number of retained modes (if no reduction then m=n).  
 Path\$ = string variable indicating the chosen method of solution.  
 [K] provides storage for the global stiffness matrix, and the static force matrix. The stiffness matrix is a square matrix with dimensions equal to GN times 3. The static force matrix is stored with the stiffness in an addition column.  
 [M] provides storage for the global mass matrix. It is a square matrix with dimensions equal to GN times 3.  
 [R] provided storage for the reduce index. It is a list of equations to be retained.  
 [P1] and [P2] are calculated and stored to for use by eigen solver in transforming eigen vectors.

#### Available Sub-Programs:

Display.Matrix  
 Store.Matrix  
 Retrieve.Matrix  
 Mat.time.Mat  
 Mat.plus.Mat  
 Invert.Matrix



```

/ +-----+
/ |      DynFEP.reduce      |
/ +-----+

```

COMMON GN,NE,DOF,n,n,PNS,Path\$

Start: a1=1: p=m: s=n-m: n1=1

```

DIM KN(n,n+1),RN(n,1),Kpp#(p,p),Kss#(s,s),Ksp#(s,p),Kps#(p,s): IF p>s THEN d=p ELSE d=s
DIM T1#(d,d),T2#(d,d) 'temporary storage
CALL Retrieve.Matrix(n,n+1,KN(),PNS+ ".K&F",a1)
CALL Retrieve.Matrix(m,a1,RN(),PNS+ ".reduce",a1)
'***** debug
CALL Display.Matrix(m,a1,RN(),"Equations to be Retained")
'*****

```

```

FOR i=1 TO m 'move equations to be retained to the top
IF i<>RN(i,1) THEN FOR j=1 TO n: SWAP KN(i,j),KN(RN(i,1),j): NEXT j 'swap row
IF i<>RN(i,1) THEN FOR k=1 TO n: SWAP K#(k,i),K#(k,RN(i,1)): NEXT k 'swap column
NEXT i

```

```

FOR i=1 TO p: FOR j=1 TO p: Kpp#(i,j)=KN(i,j): NEXT j 'build partitioned matrices
FOR k=p+1 TO n: Kps#(i,k-p)=KN(i,k): NEXT k,i
FOR i=p+1 TO n: FOR j=p+1 TO n: Kss#(i-p,j-p)=KN(i,j): NEXT j
FOR k=1 TO p: Ksp#(i-p,k)=KN(i,k): NEXT k,i
'***** debug
CALL Display.Matrix(p,p,Kpp#(),"Kpp"): CALL Display.Matrix(p,s,Kps#(),"Kps")
CALL Display.Matrix(s,p,Ksp#(),"Ksp"): CALL Display.Matrix(s,s,Kss#(),"Kss")
'*****

```

```

CALL Invert.Matrix(s,Kss#()) 'find [Kss] inverse then save partitioned matrices
CALL Store.Matrix(p,p,Kpp#(),PNS+ ".Kpp",a1): CALL Store.Matrix(p,s,Kps#(),PNS+ ".Kps",a1)
CALL Store.Matrix(s,p,Ksp#(),PNS+ ".Ksp",a1): CALL Store.Matrix(s,s,Kss#(),PNS+ ".Kss",a1)

```

```

CALL Mat.times.Mat(s,p,s,Kss#(),Ksp#(),T1#()) 'find the reduced stiffness matrix
CALL Mat.times.Mat(p,p,s,Kps#(),T1#(),T2#())
CALL Mat.plus.Mat(p,p,n1#(),Kpp#(),-n1#,T2#())
'***** debug
CALL Display.Matrix(p,p,Kpp#(),"Reduced Stiffness")
'*****
CALL Store.Matrix(p,p,Kpp#(),PNS+ ".K*",a1) 'store the reduced stiffness matrix
ERASE KN,Kpp#,Ksp#,Kps#,Kss#

```

```

DIM M#(n,n),Mpp#(p,p),Mss#(s,s),Msp#(s,p),Mps#(p,s)
CALL Retrieve.Matrix(n,n,M#(),PNS+ ".M",a1)

```

```

FOR i=1 TO m 'move equations to be retained to the top
IF i<>RN(i,1) THEN FOR j=1 TO n: SWAP M#(i,j),M#(RN(i,1),j): NEXT j 'swap row
IF i<>RN(i,1) THEN FOR k=1 TO n: SWAP M#(k,i),M#(k,RN(i,1)): NEXT k 'swap column
NEXT i

```

```

FOR i=1 TO p: FOR j=1 TO p: Mpp#(i,j)=M#(i,j): NEXT j 'build partitioned matrices
FOR k=p+1 TO n: Mps#(i,k-p)=M#(i,k): NEXT k,i
FOR i=p+1 TO n: FOR j=p+1 TO n: Mss#(i-p,j-p)=M#(i,j): NEXT j
FOR k=1 TO p: Msp#(i-p,k)=M#(i,k): NEXT k,i
ERASE M#

```

```

'***** debug
CALL Display.Matrix(p,p,Mpp#(),"Mpp"): CALL Display.Matrix(p,s,Mps#(),"Mps")
CALL Display.Matrix(s,p,Msp#(),"Msp"): CALL Display.Matrix(s,s,Mss#(),"Mss")
'*****

DIM Kpp#(p,p),Kss#(s,s),Ksp#(s,p),Kps#(p,s) 'reload the partitioned stiffness matrices
CALL Retrieve.Matrix(p,p,Kpp#(),PN$+".Kpp",a1): CALL Retrieve.Matrix(p,s,Kps#(),PN$+".Kps",a1)
CALL Retrieve.Matrix(s,p,Ksp#(),PN$+".Ksp",a1): CALL Retrieve.Matrix(s,s,Kss#(),PN$+".Kss",a1) 'recall stored [Kss] inverse

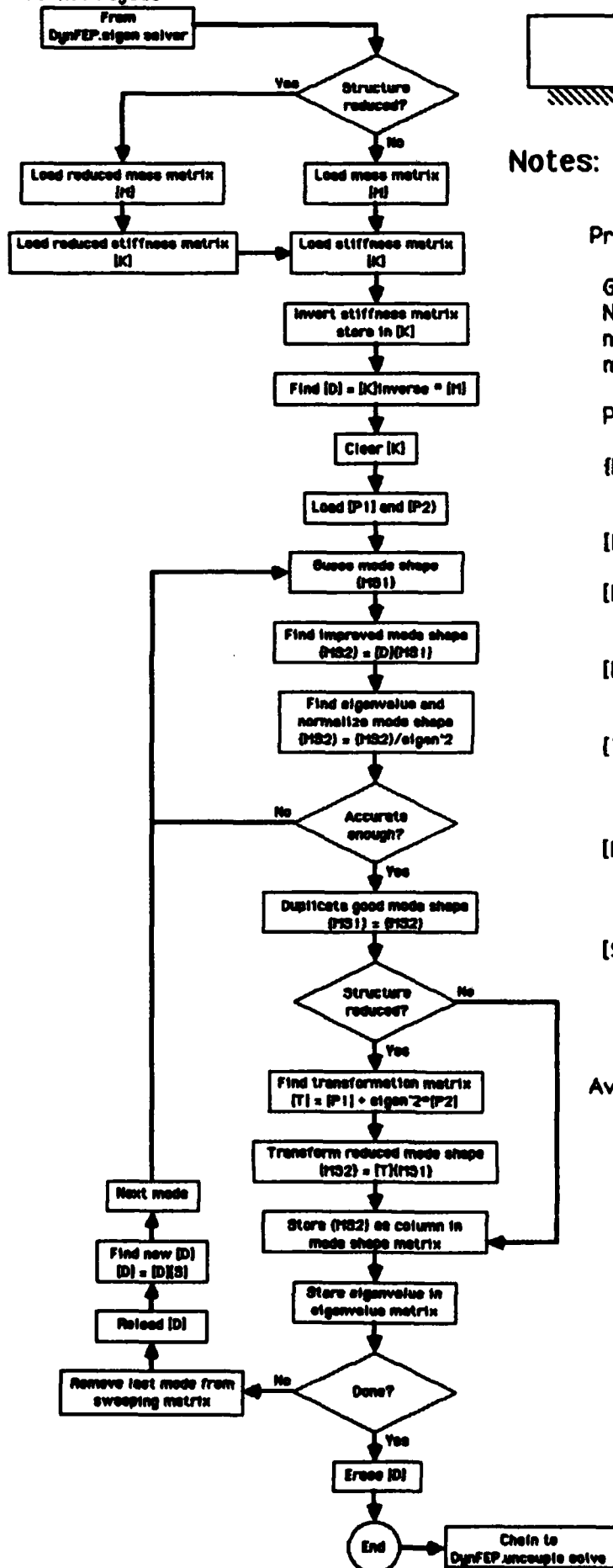
' Find the reduced mass matrix and [P1] and [P2] for use in finding [T] by DynFEP.eigen solver
FOR i=1 TO d: FOR j=1 TO d: T1#(i,j)=0: T2#(i,j)=0: NEXT j,i 'init T1# and T2#
CALL Mat.times.Mat(s,p,s,Kss#(),Ksp#(),T1#()) ' [T1] = [Kss]inv[Ksp] = [P1]
CALL Store.Matrix(s,p,T1#(),PN$+".P1",a1) 'used by DynFEP.eigen solver
CALL Mat.times.Mat(p,p,s,Mps#(),T1#(),T2#()) ' [T2] = [Mps][Kss]inv[Ksp]
CALL Mat.plus.Mat(p,p,n1#,Mpp#(),-n1#,T2#()) ' [Mpp] = [Mpp] - [Mps][Kss]inv[Ksp]
FOR i=1 TO d: FOR j=1 TO d: T2#(i,j)=0: NEXT j,i 'init T2#
CALL Mat.times.Mat(s,p,s,Mss#(),T1#(),T2#()) ' [T2] = [Mss][Kss]inv[Ksp]
CALL Mat.plus.Mat(s,p,n1#,Msp#(),-n1#,T2#()) ' [Msp] = [Msp] + [Mss][Kss]inv[Ksp]
FOR i=1 TO d: FOR j=1 TO d: T1#(i,j)=0: NEXT j,i 'init T1#
CALL Mat.times.Mat(s,p,s,Kss#(),T2#(),T1#()) ' [T1] = [Kss]inv[Mss][Kss]inv[Ksp]
FOR i=1 TO d: FOR j=1 TO d: T2#(i,j)=0: NEXT j,i 'init T2#
CALL Mat.times.Mat(s,p,s,Kss#(),Msp#(),T2#()) ' [T2] = [Kss]inv[Msp]
CALL Mat.plus.Mat(s,p,n1#,T1#(),-n1#,T2#()) ' [T1] = -[Kss]inv[Msp] + [Kss]inv[Mss][Kss]inv[Ksp] = [P2]
CALL Store.Matrix(s,p,T1#(),PN$+".P2",a1) 'used by DynFEP.eigen solver
FOR i=1 TO d: FOR j=1 TO d: T1#(i,j)=0: T2#(i,j)=0: NEXT j,i 'init T1# and T2#
CALL Mat.times.Mat(s,p,s,Kss#(),Msp#(),T1#()) ' [T1] = [Kss]inv([Msp] + [Mss][Kss]inv[Ksp])
CALL Mat.times.Mat(p,p,s,Kps#(),T1#(),T2#()) ' [T2] = [Kps][Kss]inv([Msp] + [Mss][Kss]inv[Ksp])
CALL Mat.plus.Mat(p,p,n1#,Mpp#(),-n1#,T2#()) ' [Mpp] = [Mpp] - [Mps][Kss]inv[Ksp] - [Kps][Kss]inv([Msp] + [Mss][Kss]inv[Ksp])
v[Ksp])
'***** debug
CALL Display.Matrix(p,p,Mpp#(),"Reduced Mass")
'*****

CALL Store.Matrix(p,p,Mpp#(),PN$+".M",a1) 'store the reduced mass matrix
ERASE Mpp#,Mps#,Msp#,Mss#

CLOSE: KILL PN$+".Kpp": KILL PN$+".Kps": KILL PN$+".Ksp": KILL PN$+".Kss" 'destroy temporary files
CHAIN "Basic Disk 1:DynFEP.eigen solver"
END

-----
Sub-Programs Below
-----

```



## DynFEP.eigen solver

Flow Diagram

### Notes:

#### Program Variables:

- GN = number of global nodes.
- NE = number of elements.
- n = number of unknowns in the structure.
- m = number of retained modes (if no reduction then m=n).
- Path\$ = string variable indicating the chosen method of solution.
- [MS1] and [MS2] provides storage for the mode shape vectors. The 1 and 2 refer to the and improved iterative values.
- [M] provides storage for the mass matrix. Its dimensions are mxm.
- [K] provides temporary storage of the stiffness matrix or its inverse, depending on the stage of the program.
- [D] provides storage for the result of [K]inverse \* [M], it is used to iterate toward the correct mode shape.
- [T] if the structure has been reduced, this provides storage transformation matrix to convert reduced mode shapes to full ones.
- [P1] and [P2] provide storage for matrices used in constructing the above transformation matrix, [T]. They are dimensioned (n-m)xm.
- [S] provides storage for the sweeping matrix. This matrix is used to remove last mode shape.

#### Available Sub-Programs:

Display.Matrix  
Store.Matrix  
Retrieve.Matrix  
Mat.times.Mat  
MatTrans.times.Mat  
Mat.Plus.Mat  
Invert.Mat

```

'-----
' DynFEP.eigen solver
'-----

```

COMMON GN,NE,DOF,n,m,PN\$,Path\$: GOTO Start

-----  
Subroutines Below  
-----

Remove.Last.Mode:

```

FOR i=A1 TO m: T1$(A1,i)=0: FOR j=A1 TO m: T2$(i,j)=0: NEXT j: NEXT i: T3$(A1,A1)=0 'init temp storage
CALL MatTrans.times.Mat(A1,m,m,MS1$( ),M$( ),T1$( )) ' [T1] = (MS1)tran[M]
CALL Mat.times.Mat(m,m,A1,MS1$( ),T1$( ),T2$( )) ' [T2] = (MS1)(MS)tran[M]
CALL Mat.times.Mat(A1,A1,m,T1$( ),MS1$( ),T3$( )) ' T3 = (MS1)tran[M](MS1)
aa#=1/T3$(A1,A1): CALL Mat.plus.Mat(m,m,aa$,S$( ),-aa$,T2$( ))
'CALL Retrieve.Matrix(m,m,T2$( ),PN$+".D",a3) ' load original [D] into T2#
FOR i=A1 TO m: FOR j=A1 TO m: T2$(i,j)=D$(i,j): D$(i,j)=0: NEXT j, i ' init [D]
CALL Mat.times.Mat(m,m,m,T2$( ),S$( ),D$( )) ' new[D] = original[D][S]latest
RETURN

```

Create.Eigen.Files:

```

RL=m#8: F$=PN$+".S": OPEN F$ AS #1 LEN=RL: FIELD#1,RL AS BB$ ' the shape file
a$=MKD$(0): FOR i=1 TO m: b$=b$+a$: NEXT i ' load shape file with zeros
FOR i=1 TO n: LSET BB$=b$: PUT#1,i: NEXT i
RL=8: F$=PN$+".eigen": OPEN F$ AS #2 LEN=RL: FIELD#2,RL AS CC$ ' file for eigenvalues
RETURN

```

Start: Accuracy=.01/100: a1=1: a2=2: a3= 3: a#=1: GOSUB Create.Eigen.Files

```

CALL TEXTFONT(1): CALL TEXTSIZE(9): IF m<n THEN Flag$="x" ELSE Flag$=""
DIM D$(m,m),MS1$(m,A1),MS2$(n,A1),T1$(A1,m),T3$(A1,A1)
DIM K$(m,m+1): IF Flag$="x" THEN CALL Retrieve.Matrix(m,m,K$( ),PN$+".K",a3) ELSE CALL Retrieve.Matrix(m,m+1,K$( ),PN$+".K
&F",a3)
'***** 'debug
PRINT USING "n= ## m= ## Flag= >|<";n,m,flag$
IF Flag$="x" THEN CALL Display.Matrix(m,m,K$( ),"Reduced Stiffness") ELSE CALL Display.Matrix(m,m+1,K$( ),"Stiffness")
'*****
CALL Invert.Matrix(m,K$( ))
DIM M$(m,m): CALL Retrieve.Matrix(m,m,M$( ),PN$+".M"+Flag$,a3)
'***** 'debug
CALL Display.Matrix(m,m+1,K$( ),"Inverted Stiffness")
CALL Display.Matrix(m,m,M$( ),"Mass")
'*****
CALL Mat.times.Mat(m,m,m,K$( ),M$( ),D$( )) ' [D] = [K]inv[M]
CALL Store.Matrix(m,m,D$( ),PN$+".D",a3) ' temporary file
ERASE K$ ' clear some memory
DIM S$(n,m),T2$(n,m): FOR i=A1 TO m: S$(i,i)=A1: NEXT i 'init [S] as [I]
IF n<m THEN DIM P1$(n,m),P2$(n,m): CALL Retrieve.Matrix(n-m,m,P1$( ),PN$+".P1",a3): CALL Retrieve.Matrix(n-m,m,P2$( ),PN$
+ ".P2",a3)'load P1 & P2

```

FOR eigen=A1 TO m ' begin solution

FOR i=A1 TO m: MS1\$(i,A1)=A1: NEXT i

i=2: j=eigen: Sign=-A1 ' create a first guess, should have one less sign change as eigen

WHILE j>i: FOR k=i TO n: MS1\$(i,A1)=Sign: NEXT k: Sign=-Sign: i=i+A1: WEND

Change=A1

```

WHILE Change>Accuracy
  CALL Mat.times.Mat(m,A1,m,D#(),MS1#(),MS2#())
  Freq2#=1/MS2#(A1,A1): Change=0
  FOR i=A1 TO m
    MS2#(i,A1)=MS2#(i,A1)*Freq2#
    Num=ABS(MS1#(i,A1)-MS2#(i,A1))/MS2#(i,A1): IF Num>Change THEN Change=Num
    MS1#(i,A1)=MS2#(i,A1): MS2#(i,A1)=0
  NEXT i
  ***** 'debug
  IF z=0 THEN CLS
  CALL MOVETO(2,50):z=z+1: PRINT USING "Try###";z
  PRINT Change
  'CALL Display.Matrix(n,a1,MS1#(),"Trial Vector")
  *****
WEND:z=0

IF n=m THEN FOR i=1 TO n: MS2#(i,A1)=MS1#(i,A1): NEXT i: GOTO Skip 'transform if structure not reduced
FOR i=1 TO n-m: FOR j=1 TO m: T#(i,j)=0: NEXT j,i ' store [P1] in [T]
CALL Mat.plus.Mat(n-m,m,a#T#(),Freq2#,P2#()) ' find [T] then below create matrix with [1] over -[T]
FOR i=1 TO n-m:FOR j=1 TO m:T#(i+m,j)=-T#(i,j):NEXT j,i:FOR i=1 TO m:FOR j=1 TO m:T#(i,j)=-T#(i,j):NEXT j,i
FOR i=m TO n: MS2#(i,A1)=0: NEXT i 'finish initializing MS2#
CALL mat.times.Mat(n,A1,m,T#(),MS1#(),MS2#()) ' full eigenvector stored in MS2#
Skip:
***** 'debug
PRINT USING "Mode Shape ## Freq =##.##":eigen,SQR(Freq2#)
CALL Display.Matrix(n,A1,MS2#(),"Mode Shape")
*****

'Prepare for next mode shape
Lt=(eigen-A1)*8: Rt=(m-eigen)*8
FOR i=1 TO n: GET#1,i: a$=BB$
a$=LEFT$(a$,Lt)+MKD$(MS2#(i,1))+RIGHT$(a$,Rt)
LSET BB$=a$: PUT#1,i ' store element in eigen vector matrix
NEXT i
LSET CC$=MKD$(Freq2#): PUT#2,eigen ' store square of eigenvalue
IF eigen<n THEN GOSUB Remove.Last.Mode
NEXT eigen

CLOSE: KILL P#$.D$: KILL P#$.K$: KILL P#$.M$: KILL P#$.P1$: KILL P#$.P2$ ' destroy temporary files
***** 'debug
CALL Retrieve.Matrix(n,m,S#(),P#$.S",a1)
CALL Display.Matrix(n,m,S#(),"Mode Shapes")
CALL Retrieve.Matrix(m,a1,E#(),P#$.eigen",a1)
CALL Display.Matrix(m,n1,E#(),"Eigen values")
*****
CHAIN "Basic Disk 1:DynFEP.uncouple/solve"
END

```

-----  
Sub-Programs Below  
-----

# DynFEP.uncouple/solve Flow Diagram

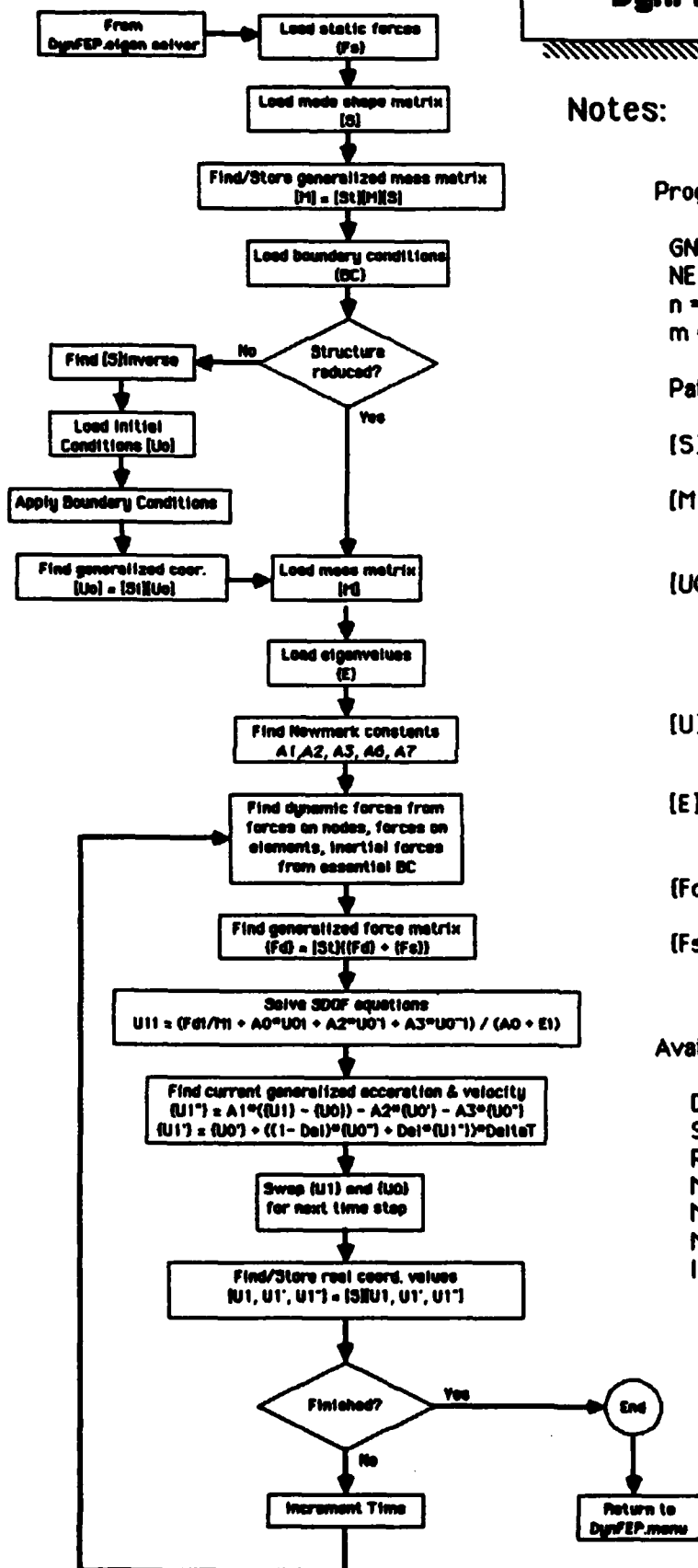
## Notes:

### Program Variables:

- GN = number of global nodes.  
NE = number of elements.  
n = number of unknowns in the structure.  
m = number of retained modes (if no reduction then  $m=n$ ).  
Path\$ = string variable indicating the chosen method of solution.  
(S) provides storage for the mode shape matrix.  
(M) provides storage for the real or generalized mass matrix, depending on the stage in the program. Its dimensions are  $m \times m$ .  
(U0) and (U1) provide storage for the current and last generalized displacement, velocity, and acceleration vectors. Treated mathematically as 3 column matrices, they are stored as a  $n \times 3$ .  
(U) provides storage for the current real displacements. Its dimensions is  $n \times 1$ .  
(E) provides storage for the eigenvalue matrix. Mathematically it's a square diagonal matrix, it's stored as a  $m \times 1$ .  
(Fd) provides storage for generalized dynamic forces. It is dimensioned as  $m \times 1$ .  
(Fs) provides storage for static forces. It is dimensioned as  $n \times 1$ .

### Available Sub-Programs:

Display.Matrix  
Store.Matrix  
Retrieve.Matrix  
Mat.times.Mat  
MatTrans.times.Mat  
Mat.Plus.Mat  
Invert.Mat



```

/ +-----+
/ | DynFEP.uncouple/solve |
/ +-----+

```

COMMON GN,NE,DOF,n,m,PN\$,Path\$: GOTO Start

Subroutines Below

ReadHistoryFile:

```

OPEN History$ AS #4 LEN=16: FIELD #4,8 AS Z$(1),8 AS Z$(2)
GET#4,1: Max=CVS(Z$(1)): Min=2: i=INT((Max-Min)/2+1)
GET#4,i: T1=CVS(Z$(1))
WHILE Max>Min+1 AND T1<>T+phase
  IF T1<T+phase THEN Min=i: i=Min+INT((Max-i)/2)
  IF T1>T+phase THEN Max=i: i=Max-INT((i-Min)/2)
  GET#4,i: T1=CVS(Z$(1))
WEND: IF T+phase=T1 THEN f=CVS(Z$(2)): GOTO found
GET#4,Min: T1=CVS(Z$(1)): f1=CVS(Z$(2)): IF T+phase=T1 THEN f=f1: GOTO found
GET#4,Max: T2=CVS(Z$(1)): f2=CVS(Z$(2)): IF T+phase=T1 THEN f=f2: GOTO found
f=(T+phase-T1)*(f2-f1)/(T2-T1)+f1 ' interpolate
found: CLOSE#4: RETURN

```

AssembleForceMat: last=0

```

FOR i=1 TO GN
  FOR j=0 TO DOF-1: index=(i-1)*DOF+j+1
    IF BC$(index,1)=1 THEN GOSUB NodeDynForces
  NEXT j,i: RETURN

```

```

NodeDynForces: k=3+5*j: Dload=0: A=1+j*4: IF last<i THEN GET#1,i: last=i 'set index's and read node dynamic forces
IF MID$(Flg1$,A+2,2)="11" THEN amp=CVS(N$(k+1)):angle=CVS(N$(k+2)):phase=CVS(N$(k+3)):Dload=amp*SIN(t*angle+phase)'Harm
Load
IF MID$(Flg1$,A+2,2)="10" THEN History$=E$(k+4):GOSUB ReadHistoryFile:Dload=f ' non-harmonic load
Fd$(index,1)=Fd$(index,1)+Dload ' add in force; positive to right, upward, clockwise
RETURN

```

InertialForces:

```

FOR i=1 TO 2: GET#1,EBC$(i,1): j=EBC$(i,2): A=3+(j-1)*4: k=3+(j-1)*5: Displ=0
IF MID$(Flg1$,A+2,2)="11" THEN amp=CVS(N$(k+1)):angle=CVS(N$(k+2)):phase=CVS(N$(k+3)):Displ=Displ+amp*SIN(t*angle+phas
e)
IF MID$(Flg1$,A+2,2)="10" THEN History$=E$(k+4):GOSUB ReadHistoryFile:Displ=Displ+f ' non-harmonic Displacement
FOR k=1 TO GN: index=(k-1)*DOF+j: Fd$(index,1)=-Displ: NEXT k
NEXT i: k=0
FOR i=1 TO GN*NE ' apply essential BC
  IF BC$(i,1)=1 THEN k=k+1: IF i<>k THEN SWAP Fd$(k,1),Fd$(i,1)
NEXT i
CALL Mat.times.Mat(n,n1,n,M$( ),Fd$( ),T2$( ))
RETURN

```

ElementMatrixAssembler:

```

FOR ELEMENT=1 TO NE
  GOSUB BuildElementMatrices
  IR=(N1%-1)*DOF:IC=(N2%-1)*DOF
  '--- Assem.Dyn.Elem.Forces:
  FOR i=1 TO DOF: Fd$(IR+i,1)=Fd$(IR+i,1)+f$(i): Fd$(IC+i,1)=Fd$(IC+i,1)+f$(i+DOF): NEXT i
  '---

```

NEXT element: RETURN

BuildElementMatrices:

```
GET#2,ELEMENT:N1%=CVI(Lt%):N2%=CVI(Rt%) ' get left and right global node #'s
GET#1,N1%:X1=CVS(N%(1)):Y1=CVS(N%(2)) ' get left side coord's
GET#1,N2%:X2=CVS(N%(1)):Y2=CVS(N%(2)) ' get right side coord's
L=SQR((Y1-Y2)^2+(X1-X2)^2) ' find element length
'--- Elem.Dyn.Forces:
DDload=0:TDload=0
IF MID$(Fig2$,2,2)="11" THEN amp=CVS(E$(10)):angle=CVS(E$(11)):phase=CVS(E$(12)):DDload=amp*SIN(t*angle+phase)'Harm
Dyn
IF MID$(Fig2$,2,2)="10" THEN History$=E$(13):GOSUB ReadHistoryFile:DDload=f ' non-harmonic dyn load
IF MID$(Fig2$,5,2)="11" THEN amp=CVS(E$(16)):angle=CVS(E$(17)):phase=CVS(E$(18)):TDload=amp*SIN(t*angle+phase)'Harm
TanDyn
IF MID$(Fig2$,5,2)="10" THEN History$=E$(19):GOSUB ReadHistoryFile:TDload=f 'non-harmonic tan dyn load
f#(1)=L*TDload/2:f#(4)=f#(1) ' positive to the right
f#(2)=L*DDload/2:f#(5)=f#(2) ' positive upward
f#(3)=-L^2*DDload/12:f#(6)=-f#(3) ' positive clockwise
'---
IF X1-X2=0 THEN angle=SGN(Y1-Y2)*Pi#/2 ELSE angle=2*Pi#-ATN((Y1-Y2)/(X1-X2))
IF angle>2*Pi#+.003 OR angle<2*Pi#-.003 THEN TransformDynForce
RETURN
```

TransformDynForce: 'Subroutine to transform Stiffness, Mass, and Force element matrices

```
'--- BuildTransformationMat: ' build [T]
FOR i=1 TO 6:FOR j=1 TO 6:TW(i,j)=0:NEXT j,i ' initialize
TW(1,1)=COS(angle):TW(4,4)=TW(1,1):TW(2,2)=TW(1,1):TW(5,5)=TW(1,1)
TW(1,2)=-SIN(angle):TW(4,5)=TW(1,2):TW(2,1)=-TW(1,2):TW(5,4)=TW(2,1)
TW(3,3)=1:TW(6,6)=1
'---
FOR i=1 TO 6:CN(i,1)=0:FOR k=1 TO 6:CN(i,1)=CN(i,1)+TW(k,i)*f#(k):NEXT k,i:FOR i=1 TO 6:f#(i)=CN(i,1):NEXT 'tran[T]*{f}
RETURN
```

Get.deltaT.and.Time.Steps:

```
DeltaT=2/SQR(EN(m,1)): TS=STR$(DeltaT): TS%=STR$(INT(SQR(EN(1,1))/DeltaT)+1) ' max deltaT and min #cycles
WINDOW 3,,(250,22)-(505,132),-4
CALL TEXTSIZE(10): CALL MOVETO(5,26): PRINT "Enter time step (max. shown)": CALL TEXTSIZE(12)
EDIT FIELD 1,TS,(5,30)-(250,45)
CALL TEXTSIZE(10): CALL MOVETO(5,61): PRINT "How many time steps?": CALL TEXTSIZE(12)
EDIT FIELD 2,TS,(5,65)-(250,80): EDIT FIELD 1
BUTTON 1,1,"OK",(200,84)-(250,102)
i=1
loop:
d=DIALOG(0)
IF d=1 OR d=6 THEN done 'got OK button or RETURN
IF d=2 THEN i=DIALOG(2): EDIT FIELD i 'got field selection
IF d=7 THEN i=(i MOD 2)+1: EDIT FIELD i 'got TAB key
GOTO loop
done: CALL TEXTSIZE(10): DeltaT=VAL(EDIT$(1)): NumSteps=VAL(EDIT$(2)): WINDOW CLOSE 3
RETURN
```

```
Start: Pi#=4*ATN(1): CR%=CHR$(13): n1=1: n3=3: n4=4: one#=1
DIM N%(17),E$(19)
F%=FN%+".nodes":OPEN F% AS #1 LEN=92
```



```

FIELD#1,12 AS Flg1$, 4 AS N$(1), 4 AS N$(2), 4 AS N$(3), 4 AS N$(4), 4 AS N$(5), 4 AS N$(6), 8 AS N$(7), 4 AS N$(8), 4 AS N$(9), 4 AS N$(10), 4 AS N$(11), 8 AS N$(12), 4 AS N$(13), 4 AS N$(14), 4 AS N$(15), 4 AS N$(16), 8 AS N$(17)
F$=PN$+".elements":OPEN F$ AS #2 LEN=94
FIELD#2,6 AS Flg2$,2 AS Lt$,2 AS Rt$,4 AS E$(1),4 AS E$(2),4 AS E$(3),4 AS E$(4),4 AS E$(5),4 AS E$(6),4 AS E$(7),4 AS E$(8),4 AS E$(9),4 AS E$(10),4 AS E$(11),4 AS E$(12),8 AS E$(13),4 AS E$(14),4 AS E$(15),4 AS E$(16),4 AS E$(17),4 AS E$(18),8 AS E$(19)
F$=PN$+".disp1": OPEN F$ AS #3 LEN=24
FIELD#3,8 AS U$, 8 AS V$, 8 AS Ac$

IF n<>m THEN Flag$="*" ELSE Flag$="" ' flag reduced structure
DIM KN(n,n+1),Fs#(n,1): CALL Retrieve.Matrix(m,m+1,KN(),PN$+".K&F",n4)
FOR i=1 TO n: Fs#(i,1)=KN(i,n+1): NEXT i: ERASE KN ' load static force matrix

'Find/Store Generalized Mass Matrix
DIM MH(n,n),Mdia#(m,n1),T1#(n,n): CALL Retrieve.Matrix(n,n,MH(),PN$+".M",n4)
DIM S#(n,n): CALL Retrieve.Matrix(n,n,S#(),PN$+".S",n4) ' load mode shapes
CALL MatTrans.times.Mat(m,n,n,S#(),MH(),T1#())
FOR i=1 TO m: FOR j=1 TO m: MH(i,j)=0: NEXT j, i ' init MH
CALL Mat.times.Mat(m,m,n,T1#(),S#(),MH()): FOR i=1 TO m: Mdia#(i,1)=MH(i,i): NEXT i ' store dia. in Mdia#
'***** debug
CALL Display.Matrix(m,m,MH(),"Generalized Mass Matrix")
CALL Display.Matrix(n,n,S#(),"Mode Shapes")
'*****
CALL Retrieve.Matrix(n,n,MH(),PN$+".M",n4) ' [M] needed to find inertial forces
ERASE T1# ' clear some memory
DIM EN(n,1): CALL Retrieve.Matrix(m,n1,EN(),PN$+".eigen",n4) ' load eigenvalues

DIM BC#(GN*DOF,1):p=GN*DOF: CALL retrieve.Matrix(p,n1,BC#(),PN$+".BC",n4) ' load boundary condition index
'***** debug
CALL Display.Matrix(GN*DOF,n1,BC#(),"Boundary Condition Index")
'*****

DIM U0#(n,3),U1#(GN*NE,3): GOTO Skip 'trouble with initial conditions file, can't resolve
IF Flag$="*" THEN Skip ' initial conditions must=0 if structure is reduced
CALL Invert.Matrix(n,n,S#())
'***** debug
CALL Display.Matrix(n,n,S#(),"Inverted Mode Shapes")
'*****
CALL Retrieve.Matrix(GN*DOF,n3,U1#(),PN$+".initial",n4): k=0
FOR i=1 TO GN*DOF 'apply boundary conditions to intial conditions
IF BC#(i,1)=1 THEN k=k+1: IF i<>k THEN FOR j=1 TO 3: SWAP U1#(k,j),U1#(i,j): NEXT j
NEXT i
CALL Mat.times.Mat(n,n1,n,S#(),U1#(),U0#()) ' find generalized initial conditions
CALL Retrieve.Matrix(n,n,S#(),PN$+".S",n4) ' reload mode shapes
Skip: 'CALL Store.Matrix(n,n3,U1#(),PN$+".disp1",n4) 'store initial conditions in displacement file

Find.Essential.BC: i=0: j=1
WHILE j<=2: i=i+1: index=(i-1)*DOF+j ' find out where uniform base movement stored
IF BC#(index,1)=0 THEN EBC$(j,1)=i: EBC$(j,2)=j: j=j+1: i=0
WEND

'Get or calculate constants
delta=1/2:alpha=1/6: GOSUB Get.deltaT.and.Time.Steps
A0=1/(alpha*DeltaT^2):A2=1/(alpha*DeltaT):A3=1/(2*alpha)-1 ' calculate constants
A6=DeltaT*(1-delta):A7=delta*DeltaT

```

```

FOR Counter=1 TO NumSteps ' begin solution loop

DynForceMat: FOR i=1 TO GN#NE: Fd#(i,1)=0: NEXT i: FOR i=1 TO n: T2#(i,1)=0: NEXT i 'init Fd# & T2#
GOSUB InertialForces: GOSUB AssembleForceMat: GOSUB ElementMatrixAssembler
  FOR i=1 TO GN#NE ' apply essential BC
    IF BC#(i,1)=1 THEN k=k+1: IF i<>k THEN SWAP Fd#(k,1),Fd#(i,1)
  NEXT i
  CALL Mat.Plus.Mat(n,n1,one#,Fd#(),one#,T2#()) 'add node/element forces and inertial forces
  CALL Mat.Plus.Mat(n,n1,one#,Fd#(),one#,Fs#()) 'add dynamic and static forces
  FOR i=1 TO n: T2#(i,1)=0: NEXT i 'init T2#
  CALL MatTrans.times.Mat(m,n1,n,SW(),Fd#(),T2#()) ' find generalized dyn. force matrix

Solve:
  FOR i=1 TO m
    U1#(i,1)=(Fd#(i,1)/Mdia#(i,1)+A0*U0#(i,1)+A2*U0#(i,2)+A3*U0#(i,3))/(A0+EW(i,1))
  NEXT i
  FOR i=1 TO m ' find U and A vectors and store displacements in U
    U1#(i,3)=A0*(U1#(i,1)-U0#(i,1))-A2*U0#(i,2)-A3*U0#(i,3)
    U1#(i,2)=U0#(i,2)+A6*U0#(i,3)+A7*U1#(i,3)
    FOR j=1 TO 3: U0#(i,j)=U1#(i,j): U1#(i,j)=0: NEXT j ' [U0] = [U1] for next time step, init [U1]
  NEXT i

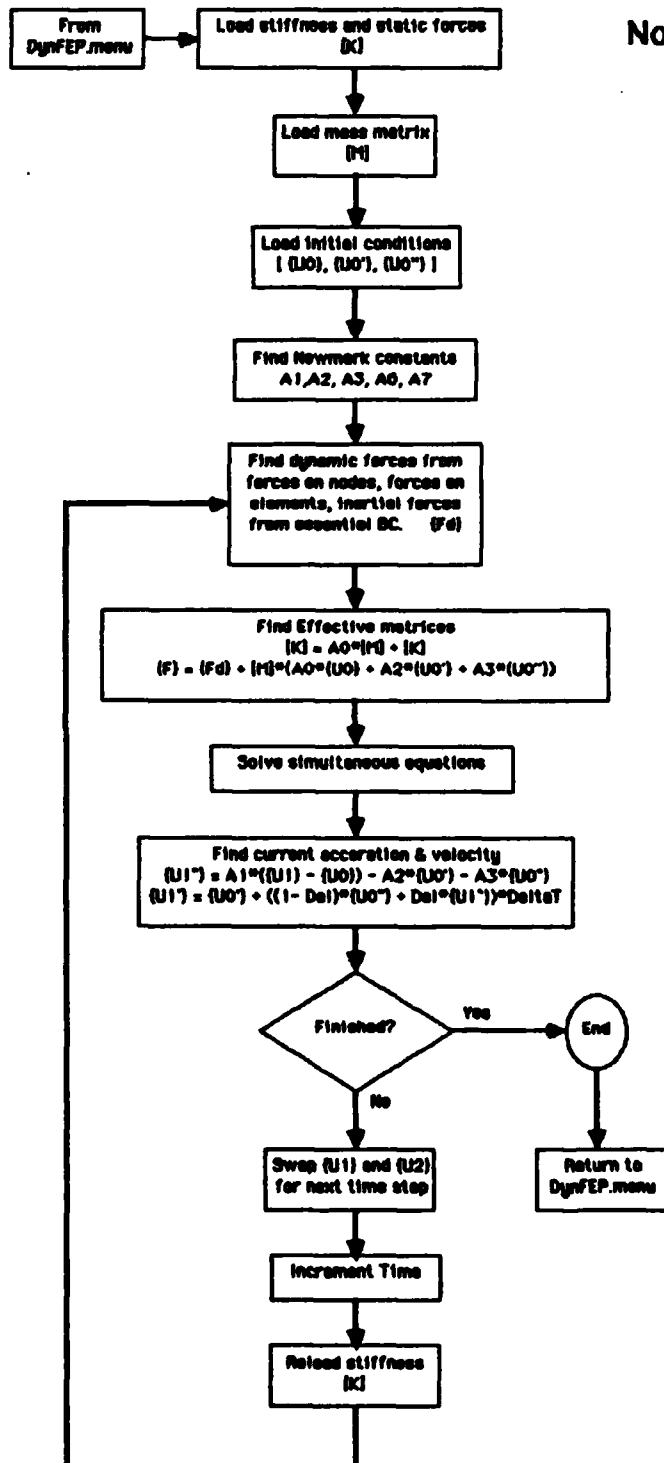
Find.Store.real.displacements:
  CALL Mat.times.Mat(n,n3,m,SW(),U0#(),U1#()) ' [U1] = [S][U0] convert from generalized coordinates
'***** debug only
  CALL Display.Matrix(n,n3,U1#(),"Displacement, Velocity, Acceleration")
'*****
  FOR i=1 TO n
    LSET U$=MKS$(U1#(i,1)):LSET V$=MKS$(U1#(i,2)):LSET Ac$=MKS$(U1#(i,3)):j=i+Counter*n:PUT#3,j ' save to disk
  NEXT i

T=T+DeltaT 'next time step
NEXT Counter

CLOSE: CHAIN "Basic Disk 1:DynFEP.menu"
END
-----
'
'                               Sub-Programs Below
'
-----

```

# DynFEP Flow Diagram



## Notes:

### Program Variables:

- GN = number of global nodes.
- NE = number of elements.
- n = number of unknowns in the structure.
- m = number of retained modes (if no reduction then m=n).
- Path\$ = string variable indicating the chosen method of solution.
- [K] provides storage for the global stiffness matrix, and the static force matrix. The stiffness matrix is a square matrix with dimensions equal to GN times 3. The static force matrix is stored with the stiffness in an addition column.
- [M] provides storage for the global mass matrix. It is a square matrix with dimensions equal to GN times 3.
- [U0] and [U1] provide storage for the current and last displacement, velocity, and acceleration vectors. Treated as 3 column matrices, they are stored as a nx3.
- {Fd} provides storage for generalized dynamic forces. It is dimensioned as mx1.

### Available Sub-Programs:

Display.Matrix  
Store.Matrix  
Retrieve.Matrix

```

/ -----+
/ |      DynFEP      |
/ -----+

```

COMMON GN,NE,DOF,n,m,PN\$,Path\$: GOTO Start

-----  
Subroutines Below  
-----

ReadHistoryFile: BUTTON 10,2

```

OPEN History$ AS #4 LEN=16: FIELD #4,8 AS Z$(1),8 AS Z$(2)
GET#4,1: Max=CVS(Z$(1)): Min=2: i=INT((Max-Min)/2+1)
GET#4,i: T1=CVS(Z$(1))
WHILE Max>Min+1 AND T1<>T+phase
  IF T1<T+phase THEN Min=i: i=Min+INT((Max-i)/2)
  IF T1>T+phase THEN Max=i: i=Max-INT((i-Min)/2)
  GET#4,i: T1=CVS(Z$(1))
WEND: IF T+phase=T1 THEN f=CVS(Z$(2)): GOTO found
GET#4,Min: T1=CVS(Z$(1)): f1=CVS(Z$(2)): IF T+phase=T1 THEN f=f1: GOTO found
GET#4,Max: T2=CVS(Z$(1)): f2=CVS(Z$(2)): IF T+phase=T1 THEN f=f2: GOTO found
f=(T+phase-T1)*(f2-f1)/(T2-T1)+f1 ' interpolate
found: CLOSE#4: BUTTON 10,1: RETURN

```

Guass: BUTTON 14,2

```

FOR i=1 TO n:M# =K#(i,i):FOR j=1 TO n+1:K#(i,j)=K#(i,j)/M#:NEXT j
FOR k=1 TO n:IF K#(k,i) THEN M# =K#(k,i):FOR j=i TO n+1:K#(k,j)=K#(k,j)-K#(i,j)*M#:NEXT j
NEXT k,i: BUTTON 14,1: RETURN

```

AssembleForceMat: BUTTON 7,2

```

FOR i=1 TO GN: GET#1,i ' read specified nodal loads
FOR j=0 TO DOF-1
  index=(i-1)*DOF+j+1:k=3+5*j
  '--- NodeDynForces:
  Dload=0: A=3+j*4
  IF MID$(Flg1$,A,2)="11" THEN amp=CVS(N$(k+1)):angle=CVS(N$(k+2)):phase=CVS(N$(k+3)):Dload=amp*SIN(t*angle+phase)'ha
  IF MID$(Flg1$,A,2)="10" THEN History$=E$(k+4):GOSUB ReadHistoryFile:Dload=f ' non-harmonic load
  Fd#(index)=Fd#(index)+Dload ' add in force; positive to right, upward, clockwise
  '---
  U1#(index,1)=0:k=1+4*j:IF MID$(Flg1$,k,1)="0" THEN U1#(index,1)=1 ' Flag essential B.C., used later
NEXT j,i: BUTTON 7,1: RETURN

```

Essential.B.C: BUTTON 13,2

```

Displ=0: Node=INT((i+2)/DOF): j=(i+2) MOD DOF: k=3+j*5: A=1+4*j:IF j=0 THEN GET#1,Node
IF MID$(Flg1$,A+1,1)="1" THEN Displ=CVS(N$(k)) 'static displacement
IF MID$(Flg1$,A+2,2)="11" THEN amp=CVS(N$(k+1)):angle=CVS(N$(k+2)):phase=CVS(N$(k+3)):Displ=Displ+amp*SIN(t*angle+phase)
IF MID$(Flg1$,A+2,2)="10" THEN History$=E$(k+4):GOSUB ReadHistoryFile:Displ=Displ+f ' non-harmonic Displacement
BUTTON 13,1: RETURN

```

ElementMatrixAssembler: BUTTON 8,2

```

FOR ELEMENT=1 TO NE
GOSUB BuildElementMatrices
IR=(N1%-1)*DOF:IC=(N2%-1)*DOF
'--- Assem.Dyn.Elem.Forces
FOR i=1 TO DOF: Fd#(IR+i)=Fd#(IR+i)+f#(i): Fd#(IC+i)=Fd#(IC+i)+f#(i+DOF): NEXT i

```

```

'---
NEXT element: BUTTON 8,1: RETURN

BuildElementMatrices: BUTTON 9,2
GET#2,ELEMENT:N1%=CVI(Lt$):N2%=CVI(Rt$) ' get left and right global node #'s
GET#1,N1%:X1=CVS(N$(1)):Y1=CVS(N$(2)) ' get left side coord's
GET#1,N2%:X2=CVS(N$(1)):Y2=CVS(N$(2)) ' get right side coord's
L=SQR((Y1-Y2)^2+(X1-X2)^2) ' find element length
'--- Elem.Dyn.Forces:
DDload=0:TDload=0
IF MID$(Flg2$,2,2)="11" THEN amp=CVS(E$(10)):angle=CVS(E$(11)):phaze=CVS(E$(12)):DDload=amp*SIN(t*angle+phase)'harm
IF MID$(Flg2$,2,2)="10" THEN History$=E$(13):GOSUB ReadHistoryFile:DDload=f ' non-harmonic dyn load
IF MID$(Flg2$,5,2)="11" THEN amp=CVS(E$(16)):angle=CVS(E$(17)):phaze=CVS(E$(18)):TDload=amp*SIN(t*angle+phase)'harm
IF MID$(Flg2$,5,2)="10" THEN History$=E$(19):GOSUB ReadHistoryFile:TDload=f 'non-harmonic tan dyn load
f#(1)=L*TDload/2:f#(4)=f#(1) ' positive to the right
f#(2)=L*DDload/2:f#(5)=f#(2) ' positive upward
f#(3)=-L^2*DDload/12:f#(6)=-f#(3) ' positive clockwise
'---
IF X1-X2=0 THEN angle=SGN(Y1-Y2)*Pi#/2 ELSE angle=2*Pi#-ATN((Y1-Y2)/(X1-X2))
IF angle>2*Pi#+.003 OR angle<2*Pi#-.003 THEN GOSUB TransformDynForce
BUTTON 9,1: RETURN

TransformDynForce: BUTTON 11,2:'Subroutine to transform Stiffness, Mass, and Force element matrices
GOSUB BuildTransformationMat
FOR i=1 TO 6:C#(i,1)=0:FOR k=1 TO 6:C#(i,1)=C#(i,1)+T#(k,i)*f#(k):NEXT k,i:FOR i=1 TO 6:f#(i)=C#(i,1):NEXT 'tran[T]*{f}
BUTTON 11,1: RETURN

BuildTransformationMat: BUTTON 12,2: ' build [T]
FOR i=1 TO 6:FOR j=1 TO 6:T#(i,j)=0:NEXT j,i ' initialize
IF angle MOD Pi#/2 THEN T#(1,1)=COS(angle) ELSE T#(1,1)=0
T#(4,4)=T#(1,1):T#(2,2)=T#(1,1):T#(5,5)=T#(1,1)
IF angle MOD Pi# THEN T#(1,2)=-SIN(angle) ELSE T#(1,2)=0
T#(4,5)=T#(1,2):T#(2,1)=-T#(1,2):T#(5,4)=T#(2,1)
T#(3,3)=1:T#(6,6)=1
BUTTON 12,1:RETURN

Get.deltaT.and.Time.Steps:
T$="Enter time step t.": TS$="Now many time steps?" ' max deltaT and min #cycles
WINDOW 3,,(250,22)-(505,132),-4: CALL TEXTFONT(1)
CALL TEXTSIZE(12): CALL MOVETO(5,26): PRINT "Enter time step (max. shown)": CALL TEXTSIZE(12)
EDIT FIELD 1,T$(5,30)-(250,45)
CALL TEXTSIZE(12): CALL MOVETO(5,61): PRINT "How many time steps?": CALL TEXTSIZE(12)
EDIT FIELD 2,TS$(5,65)-(250,80): EDIT FIELD 1
BUTTON 1,1,"OK",(200,84)-(250,102)
i=1
loop:
d=DIALOG(0)
IF d=1 OR d=6 THEN done 'got OK button or RETURN
IF d=2 THEN i=DIALOG(2): EDIT FIELD i 'got field selection
IF d=7 THEN i=(i MOD 2)+1: EDIT FIELD i 'got TAB key
GOTO loop
done: CALL TEXTFONT(4): CALL TEXTSIZE(9): DeltaT=VAL(EDIT$(1)): NumSteps=VAL(EDIT$(2)): WINDOW CLOSE 3
RETURN

BigText:CALL TEXTFONT(0):CALL TEXTSIZE(12):RETURN ' Chicago

```

```
LittleText:CALL TEXTFONT(1):CALL TEXTSIZE(9):RETURN ' Geneva
NormalText:CALL TEXTFONT(1):CALL TEXTSIZE(10):RETURN ' Geneva
FormattedText:CALL TEXTFONT(4):CALL TEXTSIZE(9):RETURN ' Monaco
-----
```

```
Start: CR%=CHR$(13): Pi%=4*ATN(1): n1=1: n3=3: n4=4: n=GN*DOF: GOSUB FormattedText
```

```
' create status windows
F$="DynFEP.info":OPEN F$ AS #1 LEN=40
FIELD#1, 2 AS X1$, 2 AS Y1$, 2 AS X2$, 2 AS Y2$, 30 AS Title$, 2 AS Type$
WINDOW 2,"DynFEP Input/Output Window", (14,61)-(512,263),1
WINDOW 1,"DynFEP Status Window", (4,41)-(424,161),1
FOR i=1 TO 10: GET#1,i
  x1=CUI(X1$): y1=CUI(Y1$): x2=CUI(X2$): y2=CUI(Y2$): A$=Title$: kind=CUI(Type$)
  WHILE RIGHT$(a$,1)="" :a$=LEFT$(a$,LEN(a$)-1):WEND:BUTTON i,1,a$,(x1,y1)-(x2,y2),kind
NEXT i:CLOSE#1
```

```
DIM K#(n,n+1),M#(n,n),Fd#(n),U0#(n,3),U1#(n,3),N$(17),E$(19)
F$=PN$+".nodes":OPEN F$ AS #1 LEN=92
FIELD#1,12 AS Flg1$, 4 AS N$(1), 4 AS N$(2), 4 AS N$(3), 4 AS N$(4), 4 AS N$(5), 4 AS N$(6), 8 AS N$(7), 4 AS N$(8), 4 AS N$(9), 4 AS N$(10), 4 AS N$(11), 8 AS N$(12), 4 AS N$(13), 4 AS N$(14), 4 AS N$(15), 4 AS N$(16), 8 AS N$(17)
F$=PN$+".elements":OPEN F$ AS #2 LEN=94
FIELD#2,6 AS Flg2$, 2 AS Lt$, 2 AS Rt$, 4 AS E$(1), 4 AS E$(2), 4 AS E$(3), 4 AS E$(4), 4 AS E$(5), 4 AS E$(6), 4 AS E$(7), 4 AS E$(8), 4 AS E$(9), 4 AS E$(10), 4 AS E$(11), 4 AS E$(12), 8 AS E$(13), 4 AS E$(14), 4 AS E$(15), 4 AS E$(16), 4 AS E$(17), 4 AS E$(18), 8 AS E$(19)
F$=PN$+".displ": OPEN F$ AS #3 LEN=24: FIELD#3,8 AS U$, 8 AS V$, 8 AS Ac$
```

```
Load.Global.Matrices: BUTTON 1,2
CALL Retrieve.Matrix(n,n+1,K#(),PN$+".K&F.c",n4)
CALL Retrieve.Matrix(n,n,M#(),PN$+".M.c",n4)
CALL Retrieve.Matrix(n,n,3,U0#(),PN$+".initial",n4)
FOR i=1 TO n 'start "displ" file 2 zero
  LSET U$=MKD$(U0#(i,1)):LSET V$=MKD$(U0#(i,2)):LSET Ac$=MKD$(U0#(i,3))
  PUT#3,i:NEXT i
BUTTON 1,0
```

```
'***** debug only
CALL DisplayMatrix(n,n+1,K#(),"Stiffness")
CALL DisplayMatrix(n,n,M#(),"Mass")
CALL DisplayMatrix(n,n,3,U0#(),"Initial Conditions")
'*****
```

```
'Get or calculate constants
delta=1/2:alpha=1/4: GOSUB Get.deltaT.and.Time.Steps
A0=1/(alpha*DeltaT^2):A2=1/(alpha*DeltaT):A3=1/(2*alpha)-1 ' calculate constants
A6=DeltaT*(1-delta):A7=delta*DeltaT
```

```
FOR Counter=1 TO NumSteps ' begin solution loop
```

```
Find.Dyn.Force.Mat: BUTTON 2,2 ' status report
GOSUB AssembleForceMat: GOSUB ElementMatrixAssembler
BUTTON 2,1 ' status report
```

```
Find.Effective.Mat: BUTTON 3,2 ' also apply BC
FOR i=1 TO N:K#(i,n+1)=K#(i,n+1)+Fd#(i): Fd#(i)=0 'add in dyn forces and init Fd# for next time step
IF U1#(i,1)<>1 THEN FOR j=1 TO N:K#(i,n+1)=K#(i,n+1)+M#(i,j)*(A0*U0#(j,1)+A2*U0#(j,2)+A3*U0#(j,3)):K#(i,j)=K#(i,j)+A0*
M#(i,j):NEXT j
```

```
IF U1$(i,1)=1 THEN GOSUB Essential.B.C: FOR j=1 TO n:K$(i,j)=-(i=j):NEXT j:K$(i,n+1)=Disp1 'set spec'd displacement
NEXT i
BUTTON 3,1 ' status report
```

```
Solve: BUTTON 4,2 ' status report
GOSUB Gauss: BUTTON 4,1: BUTTON 5,2
FOR i=1 TO n ' find V and A vectors and store displacements in U
U1$(i,1)=K$(i,n+1)
U1$(i,3)=A0*(U1$(i,1)-U0$(i,1))-A2*U0$(i,2)-A3*U0$(i,3)
U1$(i,2)=U0$(i,2)+A6*U0$(i,3)+A7*U1$(i,3)
LSET U$=MKD$(U1$(i,1)):LSET V$=MKD$(U1$(i,2)):LSET Ac$=MKD$(U1$(i,3)):j=i+Counter*n:PUT#3,j ' save to disk
NEXT i: BUTTON 5,1 ' status report
```

```
'***** debug only
CALL DisplayMatrix(n,n3,U1$(),"Displacement, Velocity, and Acceleration")
WINDOW 2: PRINT USING "Time step ### of ### .";counter,NumSteps
PRINT USING "T = ##.##" Time step = ##.##";T,deltaT: WINDOW 1
'*****
```

```
NextTimeStep: BUTTON 6,2: WINDOW OUTPUT 2 ' status report
T=T+deltaT: WINDOW 1
FOR i=1 TO n: FOR j=1 TO 3: U0$(i,j)=U1$(i,j): NEXT j,i ' initialize
CALL Retrieve.Matrix(n,n+1,K$()),PNS*=".K&F.c",n4)
BUTTON 6,1 'status report
NEXT Counter
```

```
CLOSE: WINDOW CLOSE 1: WINDOW CLOSE 2: CHAIN "DynFEP.menu": END
```

-----  
Subprograms Below  
-----

A subset of the following SUB-Programs are used in most of the DynFEP programs:

-----  
Sub-Programs Below  
-----

```
SUB Retrieve.Matrix(r,c,A#(),F$,k) STATIC
  IF UBOUND(A#,1)<r OR UBOUND(A#,2)<c THEN PRINT CHR$(7)"Fatal error!": STOP
  RL=c*8: OPEN F$ AS #k LEN=RL: FIELD#k,RL AS AA$
  FOR i=1 TO r: GET#k,i: FOR j=1 TO c
    B$=MID$(AA$,8*(j-1)+1,8): A#(i,j)=CVD(B$)
  NEXT j,i: CLOSE#k
END SUB
```

```
SUB Store.Matrix(r,c,A#(),F$,k) STATIC
  IF UBOUND(A#,1)<r OR UBOUND(A#,2)<c THEN PRINT CHR$(7)"Fatal error!": STOP
  RL=c*8: OPEN F$ AS #k LEN=RL: FIELD#k,RL AS AA$
  FOR i=1 TO r: B$="": FOR j=1 TO c
    B$=B$+MKD$(A#(i,j))
  NEXT j: LSET AA$=B$: PUT#k,i: NEXT i: CLOSE #k
END SUB
```

```
SUB Display.Matrix(Row,Col,A#(2),T$) STATIC
  CALL TEXTFONT(1): CALL TEXTSIZE(9): PRINT T$
  FOR i=1 TO Row: FOR j=1 TO Col
    PRINT USING "+#.##**** ";A#(i,j);
  NEXT j: PRINT: NEXT i: PRINT
  INPUT "Press 'RETURN' to continue";a$
END SUB
```

```
SUB Mat.times.Mat(rA,cB,cArB,A#(2),B#(2),R#(2)) STATIC
  '[A] * [B] = [R]
  'rA = #rows in [A]      cArB = #cols in [A] and #rows in [B]
  'cB = #cols in [B]      [R] is dimensioned  rA X cB
  FOR i=1 TO rA: FOR j=1 TO cB: FOR k=1 TO cArB
    R#(i,j)=R#(i,j)+A#(i,k)*B#(k,j)
  NEXT k,j,i
END SUB
```

```
SUB MatTrans.times.Mat(cA,cB,rArB,A#(2),B#(2),R#(2)) STATIC
  '[A transpose] * [B] = [R]
  'cA = #cols in [A]      rArB = #rows in [A] and #rows in [B]
  'cB = #cols in [B]      [R] is dimensioned  cA X cB
  FOR i=1 TO cA: FOR j=1 TO cB: FOR k=1 TO rArB
    R#(i,j)=R#(i,j)+A#(k,i)*B#(k,j)
  NEXT k,j,i
END SUB
```

```
SUB Mat.plus.Mat(r,c,C1#,A#(2),C2#,B#(2)) STATIC
  'C1*[A] + C2*[B] = result stored in [A]
  FOR i=1 TO r: FOR j=1 TO c: A#(i,j)=C1#*A#(i,j)+C2#*B#(i,j): NEXT j,i
END SUB
```

```
SUB Invert.Matrix(n,A#(2)) STATIC
```



```
'Takes [A] * [A]-1 = [I] AND changes TO [I] * [A]-1 = [A]-1 (based on Gauss elimination)
' [A]-1 replaces [A]
DIM I#(n,n): FOR i=1 TO n: I#(i,i)=1: NEXT i 'identity matrix
FOR i=1 TO n: m# = A#(i,i): FOR j=1 TO n: A#(i,j) = A#(i,j)/m#: I#(i,j) = I#(i,j)/m#: NEXT j
FOR k=1 TO n: IF k < i THEN m# = A#(k,i): FOR j=1 TO n: A#(k,j) = A#(k,j) - A#(i,j)*m#: I#(k,j) = I#(k,j) - I#(i,j)*m#: NEXT j
NEXT k,i
FOR i=1 TO n: FOR j=1 TO n: A#(i,j) = I#(i,j): NEXT j,i: ERASE I# 'store inverse in A#
END SUB
```

```
SUB Determinant(n,A#(2),Det) STATIC
'Uses pivital condensation to find the determinant of [A#]
Mult=1: Sign=1
WHILE n > 2
  i=1: WHILE A#(i,1)=0 AND i <= n: i=i+1: WEND 'check for zero in first column, then correct
  IF i > n THEN Det=0: GOTO Finished '1st col has all zeros
  IF i > 1 THEN Sign=-Sign: FOR j=1 TO n: SWAP A#(1,j),A#(i,j): NEXT j 'swap rows and change sign
  Mult=Mult/A#(1,1)*(n-2)
  FOR i=2 TO n: FOR j=2 TO n: A#(i,j) = A#(i,j) - A#(1,j)*A#(1,i): NEXT j,i
  FOR i=1 TO n-1: FOR j=1 TO n-1: A#(i,j) = A#(i+1,j+1): NEXT j,i
  n=n-1
WEND
Det=Sign*Mult*A#(1,1)
Finished:
END SUB
```

-----

AD NUMBER  Lynch, J M.		DATE  M.	DTIC ACCESSION NOTICE
1. REPORT IDENTIFYING INFORMATION			
A. ORIGINATING AGENCY NAVAL POSTGRADUATE SCHOOL, MONTEREY, CA 93943			
B. REPORT TITLE AND/OR NUMBER Reliability Modeling of Standby and Emergency Generating Systems			
C. MONITOR REPORT NUMBER by: J. Michael Lynch. June 1984 Texas A&M			
D. PREPARED UNDER CONTRACT NUMBER N66314-73-A-2112			<u>REQUESTER:</u> 1. Put your mailing address on reverse of form. 2. Complete Items 1 and 2. 3. Attach form to reports mailed to DTIC. 4. Use unclassified information only.  <u>DTIC:</u> 1. Assign AD Number. 2. Return to requester.
2. DISTRIBUTION STATEMENT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED.			

DTIC FORM 50  
DEC 80

PREVIOUS EDITIONS ARE OBSOLETE

**END**

**FILMED**

**10-85**

**DTIC**